Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Generation of oblique plane triangulations

Bachelor Thesis

2021
Matúš Matok

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Generation of oblique plane triangulations

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	RNDr. Ing. František Kardoš, PhD.

Bratislava, 2021
Matúš Matok

97878874

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Matúš Matok
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Generation of oblique plane triangulations
Generovanie oblique planárnych triangulácií

Anotácia: V práci budeme študovať kombinatorické vlastnosti planárnych triangulácií s vlastnosťou, že pre jednotlivé steny sú trojice stupňov incidentných vrcholov navzájom rôzne (tzv. oblique triangulácie). Je známe, že takýchto triangulácií je len konečný počet a tiež sú známe horné odhady na maximálny možný stupeň takéhoto grafu. Cieľom práce je nájsť oblique triangulácie čo najväčšieho maximálneho stupňa a vygenerovať všetky možné oblique triangulácie daného maximálneho stupňa, prehľadávaním pomocou počítača.

Vedúci:	RNDr. Ing. František Kardoš, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	28.10.2020

Dátum schválenia: 31.10.2020

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

....

študent vedúci práce

97878874

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Matúš Matok
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Generation of oblique plane triangulations

Annotation: We will study combinatorial properties of planar triangulations with the additional property that there are no two faces with the same triple of degrees of incident vertices (so-called oblique triangulations). It is known that there are only finitely many such graphs, and upper bounds on maximum degree are known as well. In this work, we aim to find oblique triangulations of large maximum degree and to generate all oblique triangulations of a given maximum degree, using a computer-assisted search.

Supervisor:	RNDr. Ing. František Kardoš, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	28.10.2020

Approved: 31.10.2020 doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

....
Student Supervisor

Acknowledgments: I would like to thank my supervisor Mr. RNDr. Ing. František Kardoš, PhD. for his enormous contribution and dedication to this work. I mostly appreciate the friendly approach and the structure of the work we have set. For all of that and much more he certainly deserves credit.

Abstrakt

Rovinná triangulácia je *oblique* ak multimnožina stupňov troch incidentných vrcholov je pre každú stenu unikátna. V tejto práci sme navrhli a realizovali algoritmus, ktorý generuje oblique rovinné triangulácie na základe obmedzenia na maximálny stupeň vrchola v hľadanom grafe. Táto práca obsahuje detailný popis zmieneného algoritmu. Výsledky práce vo forme vygenerovaných oblique rovinných triangulácií sú obsiahnuté v prílohách.

Kľúčové slová: oblique graf, rovinná triangulácia, maximálny stupeň

Abstract

A plane triangulation is *oblique* if the multiset of the degrees of the three incident vertices is unique for each triangular face. In this work, we have designed and implemented an algorithm to generate oblique plane triangulations with a given maximum vertex degree. This paper contains a detailed description of that algorithm. Results in the form of discovered oblique plane triangulations are included in the appendices.

Keywords: oblique graph, plane triangulation, maximum degree

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since human race existed, all the progress and innovation was driven by curiosity. The curiosity has brought the society to the state where it currently is. Despite curiosity is not a sufficient trait on the way to breakthroughs it might be considered the most important. Sometimes it takes luck, courage and persistence to turn curiosity into something useful. However, sometimes the curiosity might lead us nowhere. Only the time will tell, if one's effort pays off. Amongst others, the research of oblique plane triangulations might be considered as a pursuit of curiosity. The utility of this subject has not yet been figured, but one can never know what the future has in store for us. That's why it never hurts to learn more about some things. Many have researched the subject of oblique plane triangulations. Hansjoachim Walther [4], Margit Voigt [3], Jens Schreyer [2] and František Kardoš with Jozef Miškuf [1] have all done an astonishing researches on this subject. They have discovered different facts and constraints of oblique plane graphs. Although, we are only going to restrain ourselves to triangulations in this paper, we might use some of their knowledge, to help us achieve our goals. The goals are of two kinds. Firstly, we are going to try to find a complete set of oblique plane triangulations for some well defined constraints. Secondly, we are going to try to find as big oblique plane triangulation as possible in terms of its maximum vertex degree.
The paper is going to start with some specific terminology that will help us orientate in the subject. This is going to be shortly followed by an overview of facts related to the topic. After we have covered the theoretical background, we are going to describe the algorithm in detail. Having described the algorithm, we will focus on its complexity and on the ways it can be improved. With all that knowledge, we are going to present a basic overview of the implementation. At the end, we are going to present our results and achievements.

Chapter 2

Theoretical background

2.1 Terminology

The following terminology is basic theory which is essential to understand the topic. For this purpose we chose definitions used by Kardoš and Miškuf [1] and Schreyer [2] in their respective works, as well as our own definitions.

A *graph* is a pair $G = (V, E)$, where $V = V(G)$ is the set of *vertices* of the graph G and $E(G)$ is a set of 2-element subsets of V, called *edges* of G. We shall not distinguish strictly between the graph and its vertex or edge set. For example, we may speak of a vertex $v \in G$ rather than $v \in V(G)$, and so on.

Two vertices $x, y \in V$ are *joined* by an edge $e \in E(G)$ if $e = \{x, y\}$. Sometimes we write $e = xy$ or $e = yx$ instead of $e = \{x, y\}$.

If two vertices are joined by an edge, they are *adjacent*. A vertex x is *incident* with an edge e if $x \in e$. We might refer to an adjacent vertex as a neighbour for simplicity.

An *embedding* of a graph G on a surface S is a drawing of G on S without edges crossing (vertices are represented by points, edges are represented by arcs between their endvertices). In this work the surface S is a plane since we are going to generate plane oblique triangulations.

If two edges are *facially adjacent* in a plane graph, it means that they are incident to a vertex v and there is no other edge incident to v between them in the embedding. Every edge is facially adjacent to two other edges in the vertex v.

The *degree* $deg(v)$ of a vertex $v \in V(G)$ is the number of vertices adjacent to v. The maximum degree in a graph G is denoted by $deg(G)$.

Let $G = (V, E, F)$ be a 3-connected simple graph embedded into a surface S with vertex set V , edge set E and face set F. A *face* α is an $\langle a_1, a_2, ..., a_k\rangle$-face if α is a k-gon and the degrees of the vertices incident with α in the cyclic order are $a_1, a_2, ..., a_k$. The lexicographic minimum $\langle b_1, b_2, ..., b_k\rangle$ such that α is a $\langle b_1, b_2, ..., b_k\rangle$-face is called the *type* of α.

A *triangulation* is a graph in which every face is a triangle (or a 3-gon). Since this work only researches triangulations, we can abbreviate the term type of face accordingly. Therefore type of a face is a triplet $\langle x, y, z \rangle$. Since the type of a face is the lexicographic minimum, we can additionally put $x \leq y \leq z$.

Let z be an integer. We consider *z-oblique* graphs, i.e. such graphs that the number of faces of each type is at most z. If $z = 1$ (all faces are of different types), the graph is said to be *oblique*.

2.2 Theoretical findings

Firstly, we are going to cover some important Theorems which are relevant to the topic of oblique triangulations.

Theorem 2.1 (Euler's formula.) *Let G be a plane connected graph, then:*

$$|V| + |F| - |E| = 2$$

Proof: This formula is considered common knowledge in the graph theory, therefore we will not include its proof in this paper.

Lemma 2.1 *Let G be an plane triangulation, where $m = |E|$ and $f = |F|$, then:*

$$f = \tfrac{2}{3}m$$

Proof: Let l_f be the amount of edges which a face f is incident to. In a triangulation G, $l_f = 3$, $\forall f \in G$. If we sum up l_f for all faces in G, we get the following:

$$\sum_{f \in G} (l_f) = 3f$$

We need to realise that every edge is accounted in the sum twice, therefore:

$$m = \tfrac{1}{2}3f$$

which is what we wanted to prove. □

Lemma 2.2 *Let G be a graph, where $m = |E|$ then:*

$$\sum_{v \in G} (deg(v)) = 2m$$

Proof: Every edge is incident to exactly two distinct vertices, therefore if we sum the degrees of all vertices in a graph, we account for every edge twice, hence:

$$\frac{1}{2}\sum_{v \in G}(deg(v)) = m$$

□

Theorem 2.2 *Let G be a triangulation, then:*

$$\sum_{v \in G}(6 - deg(v)) = 12$$

Proof: Using Euler's formula and multiplying it by 6, where $n = |V|$, $f = |F|$ and $m = |E|$ we get the following equation:

$$6n + 6f - 6m = 12$$

According to Lemma 1.1 and multiplying its claim by six, we get $6f = 4m$. By substituting $4m$ by $6f$ in the above-written formula, we get the following:

$$6n + 4m - 6m = 12$$
$$6n - 2m = 12$$

Using Lemma 1.2, we can further exchange $2m$, getting

$$6n - \sum_{v \in G}(deg(v)) = 12.$$

Through realisation that $n = |V|$ we can modify the equation accordingly:

$$\sum_{v \in G}(6) - \sum_{v \in G}(deg(v)) = 12.$$

Finally, by merging the sums, we have proven the content of this Theorem. □

Lemma 2.3 *Let G be an oblique plane triangulation. Then:*

$$v \in G \implies deg(v) \geq 3.$$

Proof:
Let $deg(v) = 0$. Then v has no edges, hence G is not connected, which is in conflict with G being a triangulation.
Let $deg(v) = 1$. Then v has only one edge denoted as vx in a face f. If $deg(x) = 1$ then G is not a triangulation. If $deg(x) = 2$ then G must contain edge xy, where

$y \neq v$. Hence the face f, consist of at least 4 edges, which is in conflict with G being a triangulation. If $deg(x) \geq 3$ then x must be incident to edges xy and xz that are incident to face f. That implies that f contains at least 5 vertices which is in conflict with G being a triangulation.
Let $deg(v) = 2$. Then G must contain edges vx, vy. In order for G to be a triangulation it must contain edge xy twice which is in conflict with G being a plane triangulation. Moreover, both faces would be of the same type, which is in conflict with G being oblique. □

Lemma 2.4 *Let G be an oblique plane triangulation. Then G must not contain a face with type $\langle 3, 3, x\rangle$ for $x \in \mathbb{N}$.*

Proof:
Let $deg(a) = deg(b) = 3$ and $deg(c) = x, x \in \mathbb{N}, x \geq 3$. Assume that G contains face abc. Both a and b are already incident to two edges. Each of them has one edge we have not mentioned yet. Assume a is adjacent to d. That implies b is also adjacent to d, because otherwise G would not be a triangulation. Since a has no other edges, in order for G to be a triangulation, G must contain edge cd. However the same goes for b, hence G would have to contain cd twice, which is in conflict with G being a plane triangulation. Such situation would correspond to the situation in Figure LINK. □

Lemma 2.5 *Let G be an oblique plane triangulation. Then G must not contain a face with type $\langle 3, x, x\rangle$ for $x \in \mathbb{N}$.*

Proof:
Let $deg(a) = 3$ and $deg(b) = deg(c) = x, x \in \mathbb{N}, x \geq 3$. Assume that G contains face abc. In that case, a has one edge we have not mentioned denoted as ad. Since a has no other edges, in order for G to be a plane triangulation, it must contain edges bd and cd. However, $deg(b) = deg(c)$ which implies $\langle a, b, d\rangle = \langle a, c, d\rangle$ which is in conflict with G being oblique. □

Theorem 2.3 *Let G be a plane triangulation with maximum degree Δ. Let V_k be the set of vertices of degree k, $3 \leq k \leq \Delta$. Then*

$$|V_k| = \begin{cases} \lfloor \frac{(\Delta-3)(\Delta-4)}{6} \rfloor, & \text{if } k = 3 \\ \lfloor \frac{(\Delta-3)(\Delta+2)}{2k} \rfloor, & \text{if } k > 3 \end{cases}$$

Proof:
We say that a vertex v gives a *token* to a face f if v is incident to f. Firstly, we need to count how many tokens can all $v_i \in V_k$ give to faces in an oblique plane triangulation. There are following cases:

- face with a type $\langle k, k, k\rangle$. In this case, vertices from V_k give three tokens as there are three vertices of degree k incident to this face. Since there can only be one face with a type of $\langle k, k, k\rangle$, V_k gives tokens 3 tokens at most to G.

- Face with a type $\langle k, k, x\rangle$, where $x \in \mathbb{N}, x > 3, x \neq k$. In this case, vertices from V_k give 2 tokens as there are 2 vertices with type k incident to this face. There are $(\Delta - 4)$ options for x, as degrees $1, 2, 3$ and k are forbidden. Hence V_k has at most $2(\Delta - 4)$ tokens in G in faces of this type.

- Face with a type $\langle k, x, x\rangle$, where $x \in \mathbb{N}, x > 3, x \neq k$. In this case, vertices from V_k give 1 token to this type of a face, since only one vertex of degree k is incident to this face. There are $(\Delta - 4)$ options for x as degrees $1, 2, 3$ and k are forbidden, hence V_k has at most $(\Delta - 4)$ tokens in G in faces of this type.

- Face with a type $\langle k, x, y\rangle$, where $x, y \in \mathbb{N}$, $x, y > 3, x \neq y, x \neq k$ and $y \neq k$. Vertices from V_k give one token to faces of this type as there is only one vertex with a degree of k incident to this face. There are $(\Delta - 3)$ options for x as degrees $1, 2, 3$ are forbidden and $(\Delta - 4)$ options for y as degrees $1, 2, 3$ and x are forbidden. That in total is $\frac{(\Delta-3)(\Delta-4)}{2}$ types of these faces, as $\langle k, x, y\rangle$ and $\langle k, y, x\rangle$ are considered as the same type. Since there is only one token given by vertices from V_k, there are at most $\frac{(\Delta-3)(\Delta-4)}{2}$ tokens from V_k in G in faces of this type.

Now that we have summed how many vertices can be at most in G for each set of types, we need to sum this number to get the maximum amount of tokens given by V_k in G. There are two cases:

- if $k = 3$, then according to the Lemmas 2.4 and 2.5, G cannot contain faces with types in cases one through 3, hence G will at most contain $\frac{(\Delta-3)(\Delta-4)}{2}$ tokens from V_3.

- if $k > 3$, then we need to sum all above mentioned cases.

$$3 + 2(\Delta - 4) + (\Delta - 4) + \frac{(\Delta - 3)(\Delta - 4)}{2} =$$

$$= 3\Delta - 9 + \frac{(\Delta - 3)(\Delta - 4)}{2} =$$

$$= (\Delta - 3)[3 + \frac{\Delta - 4}{2}] =$$

$$= \frac{1}{2}(\Delta - 3)(\Delta + 2)$$

Now that we have accumulated the maximum number of tokens given by vertices in V_k to faces in G, we need to determine how many vertices of degree k can G contain. We need to realise that each vertex in V_k gives exactly k tokens. By dividing the maximum count of tokens by k for each V_k, we get the following:

$$|V_k| = \begin{cases} \lfloor\frac{(\Delta-3)(\Delta-4)}{2\times3}\rfloor, & \text{if } k=3, \\ \lfloor\frac{(\Delta-3)(\Delta+2)}{2k}\rfloor, & \text{if } k>3, \end{cases}$$

which is the exact content of this Theorem. □

Secondly, we are going to take a look at some more specific findings about these graphs which were published by František Kardoš and Jozef Miškuf who have managed to find upper bounds of these graphs and Hansjoachim Walther who delimited the set of vertex degrees in these graphs. František Kardoš and Jozef Miškuf restricted the size of oblique plane triangulations both by maximum degree of a vertex and maximum size of a face. Since all faces in triangulations are triangles, delimiting face degree is irrelevant in this case. Let's take a look at two Theorems that delimit vertex degrees.

Theorem 2.4 (Kardoš and Miškuf [1]) *Let G be a z-oblique graph with maximum degree deg embedded into the surface with Euler's characteristic e. Then*

$$\Delta \leq 35.7z^2 + 56.9z + 42\ln(41+21z)z - 6e + 6.$$

Especially for every oblique graph G embedded on the sphere we have $\Delta \leq 235$.

Note: As the graph is embeded in a sphere (or plane), $e = 2$.

Theorem 2.5 (Kardoš and Miškuf [1]) *Let G be a z-oblique graph embedded into the surface with Euler's characteristic e. Let $v1$ and $v2$ be the vertices with the highest degree among all vertices of the graph G and let $\Delta_i = deg(v_i), i = 1, 2$. Then*

$$\Delta_1 + \Delta_2 \leq 35.7z^2 + 60.7z + 48z\ln(41+21z) - 6e + 12.$$

For every oblique graph embedded on the sphere we have $\Delta_1 + \Delta_2 \leq 268$.

Theorem 2.6 (Hansjoachim Walther [4]) *Let G be an oblique triangulation with the degree set $M = \{m_1, m_2, \ldots, m_k\}$, then $k \leq 90$.*

It is proven that the number of oblique plane triangulations is finite [1], but is not known. All of these theoretical findings will help to narrow down the computation and we will discuss their applications in the latter parts of the work.

Chapter 3

Algorithm

As the topic of this work is to construct an algorithm which is able to find oblique triangulations, we are going to cover the basic principles of the algorithm in this chapter. We are going to list some further definitions to help to understand the process better. These definitions are not from graph theory, they are only relevant to this algorithm and its specific features.

3.1 Core principle

Let G be an oblique plane triangulation. Let G' be a connected subgraph of G. Clearly, for every vertex v of G', we have $deg_{G'}(v) \leq deg_G(v)$. Moreover, every face of G' is a union of a set of faces of G.

Let G be an oblique plane triangulation, let H be a plane graph. A *realization of H in G* is a subgraph G' of G isomorphic to H.

Let H be a connected plane graph with at least two faces. A connected subgraph H' of H is called a *parent* of H if H' is obtained from H by one of the following operations:

1. remove a vertex of degree one in H, or
2. remove an edge incident to two distinct faces of H.

If this is the case, the graph H is called a *child* of H'.

Clearly, if H has a realization in G, then any parent H' of H has one as well.

Lemma 3.1 *Let G be an oblique plane triangulation, let H' be a plane graph that has a realization in G. If H' is not isomorphic to G, then there exists a plane graph H, which is a child of H', that has a realization in G.*

Proof. Let G' be a subgraph of G isomorphic to H'. Since H' is not isomorphic to G, G' is a proper subgraph of G, therefore, there exists a vertex or an edge in $G \setminus G'$.

If G' misses some vertices of G, then since G is connected, there exists a vertex $v \in G \setminus G'$ adjacent to a vertex $u \in G'$. We can create a child H of H' by adding a new vertex of degree one adjacent to the vertex corresponding to u in H'. It is easy to see that $G' \cup \{v\}$ is a realization of H in G.

If G' misses an edge joining two vertices already present in G', say u and v, then we can create a child H of H' by adding the edge joining the vertices of H' corresponding to u and v in H'. Since we add an edge joining two vertices of a connected graph, we split a face into two distinct faces. It is easy to see that $G' \cup uv$ is a realization of H in G. $\square$

As a consequence of this lemma, we get the following observation.

Let G be an (unknown) oblique plane triangulation. Let H_0 be a graph consisting of a single triangle. Then G can be found after a finite number of steps by a non-deterministic algorithm that, at each step, transforms H_i into its child H_{i+1}, until eventually H_k is isomorphic to G for some k.

In order to make this algorithm deterministic, at each step, we ought to consider all possible children of H_i as candidates for the graph H_{i+1} – we should make a search in the (infinite) tree of all the ancestors of H_0.

However, since the target graph G is oblique, there are constraints that make the number of possible children of each node of the search tree rather limited. In the following section we describe the tools used to determine the child-parent relation more precisely.

3.2 Definitions for practical application of the core principle

In order to be able to capture more precisely the way how a graph H can be realized in an oblique plane triangulation G, we will consider objects richer than graphs: We will consider vertex-labeled graphs, and we will allow vertices to be incident with semi-edges.

A *label* of a vertex in a graph H is an integer denoted by $\lambda(v)$, which represents the degree of the target vertex corresponding to v in a realization of H in G. Clearly, $deg_H(v) \leq \lambda(v)$. If G is an oblique plane triangulation, then the label of every vertex in G is equal to its degree.

A *semi-edge* is an edge only incident to one vertex. We will only consider graphs with the following property: For every vertex v, the number of semi-edges at v is equal to $\lambda(v) - deg_H(v)$.

For a fixed embedding of a plane graph H, each semi-edge belongs to some face.

A *closed face* f is a triangle containing no semi-edges. Therefore all faces in an oblique triangulation are closed. A face which is not closed is *open*. Some open faces might turn-out non-realizable, which we will address in a future section. We will refer to closed faces with lower-case letters, e.g., f_0 is a closed face, whereas F_1 is an open one.

Let G be an oblique plane triangulation. Let H be a subgraph of G, let uv be an edge of G. There are precisely 4 ways in which uv can be present in the graph H (see Figure 3.1 for illustration):

- both u and v are present in H and the edge uv is also present in H,
- both u and v are present in H, however the edge uv is not present in H, therefore both u and v contain a semi-edge belonging to their mutual open face,
- only u $[v]$ is present in H, containing a semi edge,
- neither u nor v is present in H, therefore uv is not present in the H in any way.

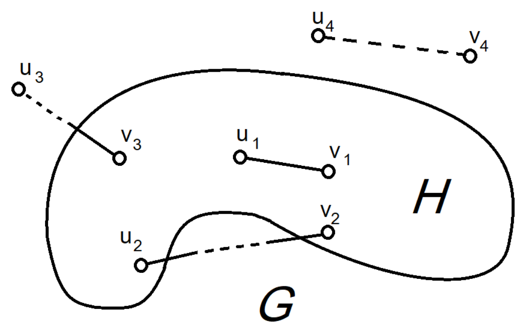


Figure 3.1: Four different ways how an edge can be present in a partial graph H contained in an oblique plane triangulation G.

3.3 Application

As mentioned above, there is a non-deterministic algorithm which transforms H_i into H_{i+1} until H_k is isomorphic to an unknown oblique plane triangulation G. As we need to make this algorithm deterministic, for every H_i we have finite amount of options how to transform it into H_{i+1}. This hints at a backtracking algorithm which will go through an infinite search tree.

Before we introduce implementation details it is important to present an overview of algorithm's structure. It is going to contain some black box functionality, which will be addressed in latter chapters of the work. The basic principle is based upon searching an infinite search tree by a recursive function. When the recursive function is called, there is an instance of a plane connected graph in a consistent state saved in a global variable and we will refer to it as H_i. In addition, our goal will be to add one closed face to the graph in each recursive call of the function.

3.3.1 Pseudo-code

recursion():

- At first we need to check if H_i is an instance of an oblique plane triangulation. It is not hard to verify if H_i is a plane triangulation. To be a plane triangulation H_i must not contain any open faces. To determine whether H_i is oblique is not difficult as we need to check if no two faces are of the same type. If it is the case, we return a copy of the graph.
- If the current instance of the graph is not an oblique plane triangulation, we do the following:
  - We initialise a result pool in which we will add all oblique plane triangulations that H_i is a realization of by collecting return values from all recursive calls on children H_{i+1} of H_i.
  - The next important task is to determine where to add the next closed face. Provided we had a non-deterministic algorithm, this step would not be necessary. Unfortunately, we do not posses such technology, therefore we need an *instruction function* that provides us with a set of instructions, so that we will not omit any child H_{i+1} that might be a realization of an oblique plane triangulation.
  - Now we have a set of instructions, which we will refer to as I and its members will be denoted as I_k. For each instruction I_k we do the following:
    - Firstly we transform H_i into H_{i+1} according to instructions in the I_k using a *transform function.*
    - After the graph is in a consistent state, we check if H_{i+1} is a potential realisation of an oblique plane triangulation using a *check function.* If it is a potential realisation then we perform a recursive call on the H_{i+1} and collect the return value into result pool.
    - At last we need to transform H_{i+1} back into H_i so that we can transform H_i into another child according to another instruction in I. For that

  purpose we will use a *inverse transform function* that will undo the transformation from the transform function.
- – Lastly we return the result pool which contains all the oblique plane triangulations that H_i is a realisation of.

After a very brief observation, it is apchild that the way this algorithm goes through the search three is a variation of a depth-first search algorithm. This raises a question of why the algorithm is not implemented as a breadth-first search. To perform a BFS it would require to store many instances of plane connected graphs which would be very memory inefficient solution for bigger inputs.

It is important to mention how the whole process begins. In other words, we need to address how we get in the initial state H_0 of the graph at which the first call of the recursive function is performed. According to the input parameters of the program, we create the first graph H_0 as follows:

- We add first three vertices a, b, c
- we add edges ab,bc,ca creating a triangle in a way that all semi-edges are in one of the two newly created faces F_0. We will refer to the other one as f_0

This is the initial state H_0. We can see that there are two distinct faces. The face F_0 is the first open face, as it contains at least one semi-edge. On the other hand, f_0 is the first closed face as it is a triangle and contains no semi-edges. Type of f_0 is $\langle\lambda(a), \lambda(b), \lambda(c)\rangle$.

There is another aspect of this algorithm worth addressing in this section. From the pseudo-code we can see that a check function is called before entering the recursion. It might be more intuitive to perform this check at the beginning of the recursive call. On the contrary, if we perform the check function before the recursive call, we have all the information of how H_i changed during transformation into H_{i+1}. Since we know that H_i was a consistent instance of potential realization of an oblique plane triangulation, it allows the check function to only focus on what has changed, whereas if we performed the check function in the beginning of the recursion it would require to check the entire graph or send the information by a complicated set of parameters.

That leaves us with 3 functions that we need to describe:

1. instruction (or choose) function
2. check function
3. transform function

3.4 Choose function

The importance of the choose function stands in providing the algorithm a set of instructions of how to transform H_i into its child so we do not except any child that could be a realisation of an oblique plane triangulation. The set of instructions contains elementary instructions where each stands for a distinct child of H_i. Each instruction contains positional information and procedural information. In other words, what needs to be done and where. We will discuss the procedural part in the section 3.5. In this section we will discuss the positional part of the instruction. In order to understand the process of choosing the position, we need to introduce some additional terminology.

3.4.1 OP-transformations

An OP-transformation is a process in which a graph H_i, which is not isomorphic to any oblique plane triangulation, is transformed into H_{i+1}. It might be intuitive that operands of an OP-transformation are two distinct vertices and a face, however that is not the case. The operands of an OP-transformation are two distinct semi-edges. If both semi-edges x, y, where x is incident to a and y to b are present in H then they are merged into one edge ab. However if one of them is not present in H then the vertex it is incident to is added to the graph H and they are merged into an edge ab. If neither of them is present in H_i, then this is not a valid OP-transformation as H_{i+1} would not be connected.

An OP-transformation in a plane connected graph G is denoted by $\Theta(x, y)$, where x, y are semi-edges incident to vertices a, b respectively. We can represent the OP-transformation as a function as follows

$$\Theta(x,y) = \begin{cases} \{\}, & \text{if } x, y \text{ do not belong to the same face} \\ \{\}, & \text{if } x, y \text{ are incident to the same vertex} \\ \{\}, & \text{if G contains edge } ab \\ \{H_{i+1}\}, & \text{otherwise} \end{cases}$$

Let's denote a function Φ as

$$\Phi(M) = B,$$

where M is a set of connected plane graphs and B is a set of oblique plane triangulations. Additionally, if $H \in M$ is a realisation of an oblique plane triangulation G, then $G \in B$.

As we do not know which operation to perform so that the child of H_i is a realisation of an oblique plane triangulation, we seemingly have to perform the OP-transformation for all 2-sets of semi-edges. However that is not the case as proven in the following Lemma.

Lemma 3.2 *Let H be a plane connected graph, b be a vertex not present in H_i, S be the set of all semi-edges in $H_i \cup \{b\}$ and c a semi-edge in H_i Then*

$$\Phi(\bigcup_{x,y \in S} \Theta(x,y)) = \Phi(\bigcup_{y \in S} \Theta(c,y)).$$

In other words, it is sufficient to choose one semi-edge x from H_i as a constant parameter for Θ instead of all 2-sets of semi-edges in H_i without affecting the set of oblique plane triangulations that have a realisation in H_i.

Proof:

$\supseteq$:

Since $c \in S$:

$$H_{i+1} \in \bigcup_{y \in S} \Theta(c,y) \implies H_{i+1} \in \bigcup_{x,y \in S} \Theta(x,y),$$

which implies

$$\Phi(\bigcup_{x,y \in S} \Theta(x,y)) \supseteq \Phi(\bigcup_{y \in S} \Theta(c,y)).$$

$\subseteq$:

If H_i is not a realisation of any oblique plane triangulation, then

$$\Phi(\bigcup_{x,y \in S} \Theta(x,y)) = \{\},$$

therefore by restricting c to be a constant, we do not alter the result set of the Φ function.

Let us assume that

$$\Phi(\bigcup_{x,y \in S} \Theta(x,y)) \nsubseteq \Phi(\bigcup_{y \in S} \Theta(c,y)),$$

therefore there is a graph $H_{i+1} \in \bigcup_{x,y \in S} \Theta(x,y)$ isomorphic to a G'_j which is a realization of an oblique plane triangulation G_J does not belong to $\Phi(\bigcup_{y \in S} \Theta(c,y))$. If G_j does not belong to $\Phi(\bigcup_{y \in S} \Theta(c,y))$, then it must contain the semi-edge c, which is in conflict with G_j being an oblique plane triangulation.

Therefore

$$\Phi(\bigcup_{x,y \in S} \Theta(x,y)) = \Phi(\bigcup_{y \in S} \Theta(c,y))$$

$\square$

We will refer to c as a *primary operand* or *primary semi-edge.*

Although this section's goal was to explain where exactly the transformation is going to take place, we omitted it for a good reason. The choice of primary semi-edge is deeply connected to the complexity of the algorithm and we will address that in the future chapter. For the following sections it is sufficient to know that any primary semi-edge will do.

3.5 Transform function

As mentioned in the section 3.2 there are precisely 4 ways how an edge can be present in H_i which is a subgraph of an oblique plane triangulation G. The first option is off the table when it comes to deciding how to transform H_i to H_{i+1} as the edge is already present in H_i which is essentially our goal for all edges. The same goes for the last one as connecting two vertices which are not present in H_i would mean that H_{i+1} would not be connected. Therefore our attention comes to the second case where vertices u and v are present in H_i but the edge uv is not and third case where a vertex u is present in H_i but v is not. If we knew G then the edge uv could get into state in which it is in H_i by removing the edge uv from the graph and removing either one of the vertices respectively. Since we are trying to create G, we need to create operations that are inverse to removal of an edge or a vertex.

Before we introduce the operations, we need to differentiate between two types of situations that can occur:

- *a deterministic situation*, which means we only have one option as how to continue in the algorithm
- *a non-deterministic* or *branching situation* in which we have several options how a child of H_i can look like

3.5.1 Deterministic situation

Lemma 3.3 *Let H be a plane triangulation and ax, bx be edges. If ax is facially adjacent to bx then H must contain edge ab, which is facially adjacent to ax and bx.*

Proof:
Assume H does not contain edge ab. Then ax would be facially adjacent to bx and ay, while bx would be facially adjacent to ax and bz. That would mean that H contains a face which is at least a 5-gon, which is in conflict with H being a plane triangulation.
□

Let H_i be a plane connected graph, F_j be an open face and a, b, c are vertices incident to F_j. Additionally b has no semi-edges in F_j, whereas b and c have at least one. Lastly F_j is incident to edges ab and bc. This situation corresponds to the situation on the left picture in Figure 3.2.

Since b has no semi-edges in the face F_j it implies that ab and bc are facially adjacent. By Lemma 3.3, which implies that there must be an edge ac which is facially adjacent to ab and bc. That state corresponds to the situation on the right picture in Figure 3.2.

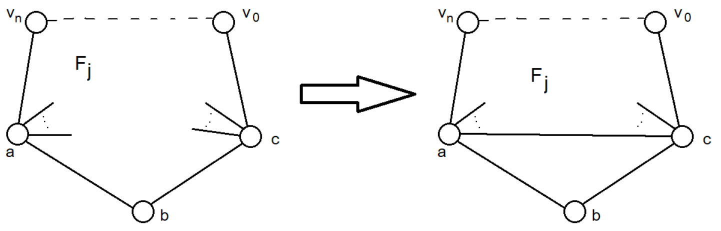


Figure 3.2: If a vertex b has no semi-edges in an open face F_j (left), then the existence of the edge ac is forced (right).

Let's take the semi-edge incident to a which is also facially adjacent to ab and denote it as s. We can consider s to be the primary operand for the OP-transformation. The other operand is also determined and it is the semi-edge incident to c which is also facially adjacent to bc. Since we only have one option for the secondary operand we call this situation deterministic.

3.5.2 Non-deterministic situation

When H is in a non-deterministic situation, there are no vertices in H that would have no semi-edges in any open face of H. In that situation, we are provided with one semi-edge c from the choose function, and we need to assert that we have a set of operations that meet the conditions of Lemma 3.2. Therefore we have two distinct sets of secondary semi-edges:

- semi-edges which are present in H,
- semi-edges which are not present in h.

The content of the set which contains secondary edges that are present in the H is quite intuitive. It is going to consist of all semi-edges of other vertices in the same open face. However, it might not be clear for the second set. Therefore, we propose following Lemma.

Lemma 3.4 *A set of secondary semi-edges which are not present S in H consist of one semi-edge for every distinct vertex label contained in a target oblique plane triangulation.*

Proof:
Let U be a set of vertices that are not present in G, but are present in the target oblique plane triangulation. Each vertex $v \in G$ has $\lambda(v)$ semi-edges, but no edges in H. Therefore the only way to differentiate between them is by their label. That implies that it is sufficient to consider semi-edges of one vertex for each label to be present in S. As mentioned, each vertex in U has no edges, which are present in H, therefore we cannot differentiate between semi-edges of a specific vertex, therefore S will consist of one semi-edge for each distinct label present in a target graph. □

3.5.3 Graph operations

Now that we have shown possible situations and what the sets of operands look like for each of them, it is about the time to introduce the actual operations. It turns out that for practical application, working strictly with semi-edges is difficult and confusing. For that reason, these operations are designed to provide a more comfortable interface. Let H_i be a plane connected graph.

- *Deterministic connect.* This operation takes 2 arguments:
  - vertex c,
  - open face F.

  As the name suggests, this operation is used in deterministic situation. Therefore, c is the vertex that has no semi-edges in face F. That determines the neighbours of c, denoted as a and b. The fact that we are in a deterministic situation also determines the primary and secondary semi-edge. Without loss to generality, let a_x, which is facially adjacent to ac be the primary semi-edge and b_x, which is facially adjacent to cb be the secondary edge. Then

$$a_x, b_x \in H_i \implies H_{i+1} \in \Theta(a_x, b_x).$$

- *Add.* This operation takes 4 arguments:
  - vertex a,
  - vertex b,
  - vertex c,
  - open face F.

  This operation is used in a non-deterministic situation when the set of secondary semi-edges consists of semi-edges that are not present in H_i. For a reason we will elaborate on later, F must contain the edge ab and c is not present in H_i. Then H_{i+1} will also contain c. Let a_x be a facially adjacent semi-edge to ab incident to vertex a and c_x be any semi-edge incident to c. Then

$$H_t \in \Theta(a_x, c_x).$$

As we have set a goal to add one closed face in each step of recursion, forcing ac to be facially adjacent to ab comes for a reason. According to claims in Lemma 3.3, H_{i+1} must contain edge cb which is facially adjacent to ab and ac, resulting in a closed face abc. Let b_y be a semi-edge incident to ab and c_y be a semi-edge facially adjacent to ac so that abc is a closed face. Then

$$H_{i+1} \in \Theta(b_y, c_y).$$

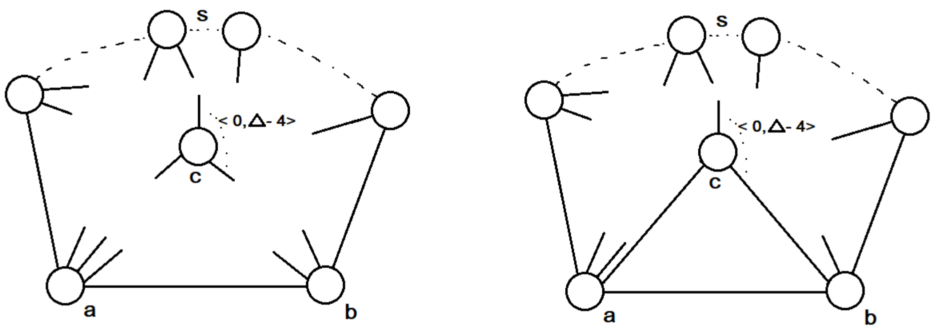


Figure 3.3: Situation before operation Add and after operation Add for parameters a, b, c, F

- *Connect.* This operation takes 5 arguments:
  - vertex a,
  - integer x_a,
  - vertex b,
  - integer x_b,
  - open face F.

This operation is used in a non-deterministic situation when the set of secondary operands is present in H_i. The integers x_a and x_b are going to be referred to as *offset* in a and b respectively. In general, a and b have x and y semi-edges in F respectively. Therefore x_a and x_b must be from *zero* to $x-1$ and $y-1$ respectively.

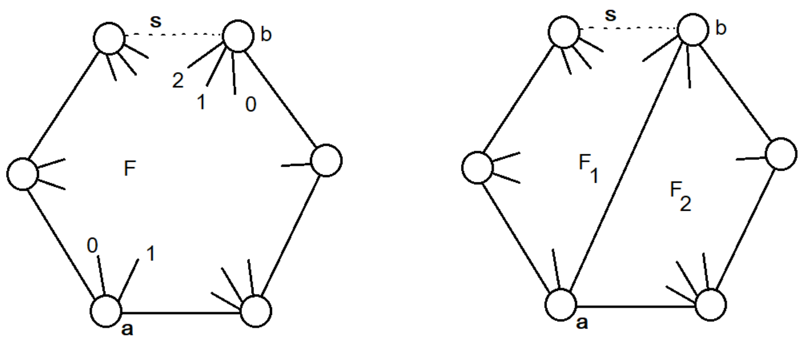


Figure 3.4: Situation before operation Connect and after for parameters $a, 1, b, 1, F$.

If not, then they are not consider as valid arguments for the operation. For a given constant embedding and convention in order of edges of a vertex, offsets determine which semi-edge of the vertices are the operands. Let a_x be the x_a^{th} semi-edge of a and b_x be the x_b^{th} semi-edge of b in F. Then

$$a_x, b_x \in H_i \implies H_{i+1} \in \Theta(a_x, b_x).$$

As a result H_{i+1} contains two open faces F_1, F_2. We know that *Add* and *Connect* must have the same semi-edge as a primary operand. We will consider a_x^{th} semi-edge of a as a primary operand. Since it must be facially adjacent to one edge of a in face F, that means that a_x can either be equal to *zero* or $x - 1$. That implies that a must contain no semi-edges in either F_1 or F_2. Without loss to generality, H_{i+1} must contain edges ab an av in F_1. Then, according to claims in Lemma 3.3 H_{i+2} must contain edge bv facially adjacent to ab and av.

For decisions made during the implementation, mainly for clarity of the algorithm, we have decided to perform these operations the way they are described above. That means, that the function "add" adds two edges in the graph adding a closed face, while "connect" only adds one, not adding a closed face directly. However, it forces H_{i+1} into deterministic situation which guarantees adding a closed face in the following recursive call.

As the algorithm works on a principle of depth-first search, we also need operations that transform H_{i+1} back to H_i. There are similarly three operations, where each of them is inverse to one of the above mentioned operations. By remembering operands for a given operation at each point of recursion, we can easily undo the changes in H_{i+1} and transform it back to H_i. For that reason it is not necessary to go into the details of these operations, as their function should be very intuitive.

3.6 Check function

Once that we have transformed H_i into H_{i+1}, we must verify whether H_{i+1} is a possible realization of an oblique plane triangulation. There are some criteria, based on which H_{i+1} can be claimed as non-realizable. Unless H_{i+1} matches any criteria, it is claimed as potentially realizable and recursive call on H_{i+1} is performed. The aforementioned order of operations in which we perform the check function before the recursive call allows us to only check the parts of the graph that have actually changed.

3.6.1 Oblique criterion

According to the basic definition of an oblique plane triangulation, there must not be any faces with the same type. If we added a face, we know its type. We need to check if H_i does not contain such face already. If H_i does contain face of the same time, we claim H_{i+1} to be non-realizable. If it does not, we perform a recursive call on H_{i+1}.

3.6.2 Non-closable face criteria

There are several situations in which we can tell that H_{i+1} is non-realizable based on the state of its face. We will say that face F is non-closable if there is no way of closing it using semi-edges in that face. Following Lemmas are going to set the conditions based on which we can claim F to be non-closable.

Lemma 3.5 *Let F be an open face in a plane connected graph H_{i+1}. Let ab be an edge incident to F, where a and b have no semi-edges in face F. Then F is non-closable.*

Proof:
Since F is incident to ab, it is incident to both a and b. Without loss to generality, F must also contain edge bc, where $c \neq a$. According to the claims of Lemma 3.3 H_{i+1} must also contain edge ac. However, a has no semi-edges in F, therefore F is non-closable. □

Lemma 3.6 *Let F be an open face in a H_{i+1} incident to vertices a, b and c and no other vertices (F is a triangle). If b has no edges in H_{i+1}, but a and c do have at least one, then F is non-closable.*

Proof:
Since F contains vertices a, b and c it must contain edges ab, bc and ca. Because b has no semi-edges in face F then according to the claims of Lemma 3.3, H_{i+1} must contain edge ca, which H_{i+1} already contains and it is not facially adjacent to neither ab nor bc. If H_{i+2} contained edge ca which is facially adjacent to ab and bc that would be in conflict with H_{i+1} not containing any multiple edges. □

Chapter 4

Complexity

In the previous chapter we have described the basic principle of the algorithm. Only using what we have covered so far, the algorithm would be perfectly functional, capable of finding any targeted oblique plane triangulation. However, it might take horrendous time to get the result. In this chapter, we are going to cover ways by which we made the algorithm's complexity much more acceptable.

There are two aspects of the search tree that we can measure. They are the depth and width. In order to improve algorithm's complexity, we have to focus on both of these aspects. Firstly, we are going to take a look, how we can restrict the depth of the search tree.

Let the maximum degree Δ of a plane triangulation G be fixed. As we've seen in Theorem 2.3, for every degree k, $3 \leq k \leq \Delta$, there can be at most a quadratic number of vertices of degree k. Therefore, the overall number of vertices in G is in $O(\Delta^3)$. Similarly, there are $O(\Delta^3)$ possible types of faces in G. Altogether, the depth of the search tree is bounded by $O(\Delta^3)$. It is very difficult to estimate the width of the tree as each plane connected graph H_i may have variable amount of children H_{i+1}. If we only were to perform operation "add", the amount of H_{i+1} for each H_i would be $O(\Delta)$. However, that is not the case as we also perform operation "connect". A very rough estimate can be made, as the width might be $O(\Delta^2)$. Then, the size of the tree can be estimated as $width^{depth}$ which is $O(e^{2\Delta^3 \log \Delta})$. If we assumed that the complexity of transformation from H_i to H_{i+1} is $O(t(\Delta))$ then the final complexity would be equal to $O(t(\Delta)e^{2\Delta^3 \log \Delta})$

4.1 Restricting depth

Firstly, it is very essential to explain why we set a goal of adding one closed face for every step of the recursion. Without doing so, we cannot really assert that the search tree would be finite, not containing any infinite branches. By adding a closed face in each step of recursion, we assert that no such branch can exist. The reason is very simple. As we know, the amount of face types is finite once the maximum degree Δ of a vertex is fixed. By adding a closed face, we are reducing the amount of faces that are not yet used in the graph. At some point, we will be in a situation of adding a face with type a. The more closed faces H_i already contains, the more likely it is for a face with type a to be already present in H_i. In that case, that branch is finished, as this a graph containing more faces with type a is in conflict of being an oblique plane triangulation.

Let G be a target oblique plane triangulation, where $\forall v \in G\ deg(g) \leq \Delta, \Delta \in \mathbb{N}$. Let $n_k = |\{v | v \in G, \delta(v) = n\}|$. Then according to Theorem 2.2, we get

$$\sum_{v \in G} (6 - deg(v)) = 12,$$

$$\sum_{k \in \mathbb{N}, k \geq 3} \Big(\sum_{v \in G, deg(v) = k} (6 - k)\Big) = 12,$$

$$\sum_{k \in \mathbb{N}, k \geq 3} (n_k (6 - n)) = 12.$$

By expanding the sum, we get

$$3n_3 + 2n_4 + n_5 - n_7 - 2n_8 - \cdots - (\Delta - 6)n_\Delta = 12,$$

$$3n_3 + 2n_4 + n_5 = 12 + n_7 + 2n_8 + \cdots + (\Delta - 6)n_\Delta.$$

According to the Theorem 2.3, we can easily calculate n_k for each k.

The combined knowledge of these two Theorems provides us with tools for two important improvements. Both of these improvements are going to extend the choose function.

- Firstly, we will check if the amount of vertices with label k has not exceeded n_k in H_{i+1}. By performing the check function before recursive call on H_{i+1}, we can only do this part of check function after operation "add", furthermore only checking if the inequality holds for k equal to the label of newly added vertex. If it holds, then H_{i+1} is realizable, otherwise it is non-realizable.

- Secondly, we will only be interested in one inequality of the two provided above:

  $$3n_3 + 2n_4 + n_5 \geq 12 + n_7 + 2n_8 + \cdots + (\Delta - 6)n_\Delta.$$

  We can easily calculate the sum on the left side of the inequality.

  $$3\lfloor \frac{(\Delta-3)(\Delta-4)}{6} \rfloor + 2\lfloor \frac{(\Delta-3)(\Delta+2)}{8} \rfloor + \lfloor \frac{(\Delta-3)(\Delta+2)}{10} \rfloor \geq 12 + n_7 + 2n_8 + \cdots + (d-6)n_d.$$

Let $m = \lfloor\frac{(\Delta-3)(\Delta-4)}{6}\rfloor + 2\lfloor\frac{(\Delta-3)(\Delta+2)}{8}\rfloor + \lfloor\frac{(\Delta-3)(\Delta+2)}{10}\rfloor$ and $l_k = |\{v \in H_{i+1}, \lambda(v) = k\}|$. Then, we need to perform following check after every operation "add". If

$$m < 12 + l_7 + 2l_8 + \cdots + (\Delta - 6)l_\Delta,$$

then H_{i+1} is non-realizable. Otherwise, H_{i+1} is realizable.
The importance of this improvement grows with Δ, as those graphs are more likely to contain vertices with bigger labels, hence adding more weight on the right side of the inequality.

4.2 Restricting width

In order to restrict the width of the search tree, it is necessary to look at how many children H_{i+1} there are for a certain plane connected graph H_i. As mentioned in the previous chapter, the choice of the primary semi-edge is crucial for the overall complexity of the algorithm. Firstly we need to remind that there are two options:

- if H_i is in a deterministic situation, there is only one child H_{i+1}.
- if H_i is in a non-deterministic situation it is very hard to estimate how many children it will have in general. That is why we will focus on it in the following text.

4.2.1 Edge type priority

The amount of children in a non-deterministic situation is a sum of children created through both operations. Therefore, we need to somehow focus on balancing these two, as both of them have to be performed with the same primary semi-edge to assert that we will not leave out any oblique plane triangulation.
In order to limit the amount of children of H_i after operation "connect", it is very tempting to choose a semi-edge in a face containing the smallest amount of semi-edges. As it turned out during our experiments, it was not the way to go. Even though the smallest face in terms of the amount of incident semi-edges might produce the smallest amount of children H_{i+1}, the branches from the operation "add" might eventually bring the formerly smaller face to the size that exceeds the size of the formerly bigger face in H_i. On the other hand, if we perform the operation "connect" on a semi-edge from a bigger face, it on one hand produces many children, but on the other hand it splits the face into two smaller faces that will not produce so many children through operation "connect". As described, it is very questionable approach to base our choice of the primary semi-edge on the size of the face it is incident to. Therefore, we will focus on restricting the amount of children after operation "add". In order to restrict

the amount of children after operation add, we will need a following definition.
An *open edge* is an edge xy that is incident to at least one open face. A *type* of an open edge is a pair $\langle\lambda(x), \lambda(y)\rangle$. The *set of potential incident closed faces* is a set of admissible face types that are not present in H_i and contain degrees $\lambda(x), \lambda(y)$. In order to reduce the amount of children after operation "add", we will choose a semi-edge that is facially adjacent to an open edge with the smallest set of potential incident faces.

4.2.2 Semi-edge driven decision

Secondly we need to highlight the importance of deterministic situation. If H_{i+1} is forced into a deterministic situation, where edges ab and bc are facially adjacent and both are incident to an open face, then the amount of available faces for open edges with types of $\langle\lambda(a), \lambda(b)\rangle$, $\langle\lambda(b), \lambda(c)\rangle$ and $\langle\lambda(c), \lambda(a)\rangle$ decreases by one, or H_{i+1} can be claimed as non-realizable if face with a type $\langle\lambda(a), \lambda(b), \lambda(c)\rangle$ is already present in H_{i+1}. Therefore forcing the children into a deterministic situation when choosing the primary semi-edge is also very beneficial. We will use this method for two situations described in the following subsections.

One-one edges

If H_i contains a face F incident to an edge ab, where both a and b have exactly one semi-edge in face F, we will nickname ab as an *one-one edge.*

Lemma 4.1 *Let H_i be a plane connected graph. If H_i contain an one-one edge ab incident to face F, then any non-deterministic operation with a semi-edge facially adjacent to ab in face F as a primary operands will force H_{i+k} into deterministic situation that will result in 3 closed faces added in H_{i+k+1}.*

Proof:
If H_i contains an one-one edge then irregardless of the operation, both a and b will have zero semi-edges in an open face in some H_{i+k}.

- If H_{i+1} was created by operation "connect", then a will have zero semi-edges in both newly created open faces. That forces H_{i+1} and H_{i+2} into deterministic situation as both will have to perform the "deterministic connect" in one of the new open faces. After both of these operations are done, then also b will inevitably contain zero semi-edges in one of the new open faces, hence H_{i+3} will also be forced into deterministic situation.
- If H_{i+1} was crated by operation "add", that has already added one new closed face and one new vertex into H_{i+1}. Similarly as previously, both a and b will

contain zero semi-edges in an open face (this time, it is the same open face F), which forces H_{i+1} and H_{i+2} into deterministic situations. □

Therefore, if we prioritise one-one edges, we can force adding 3 closed faces for each child of H_i created by a non-deterministic operation.

Semi-edge abundance as secondary criterion

We have discussed how beneficial it is to force H_{i+1} into a deterministic situation. In general, there can be more open edges with the same type in the graph. If we were to choose based only on the edge type priority, we would have to choose the specific semi-edge at random. However, there is an option to introduce a secondary criterion when choosing a specific semi-edge. In order to achieve a deterministic situation we need to complete all semi-edges incident to a vertex in a face. Therefore, the less semi-edges incident to a face vertex has, the sooner it will result in a deterministic situation. Using this logic, the secondary criterion, when choosing a specific semi-edge amongst those that are facially adjacent to an edge of the most prioritised type, will consist of picking the semi-edge that is incident to a vertex with the lowest number of semi-edges incident to the face that the open edge is incident to.

4.3 Avoiding multiple isomorphic results

So far, there is no mechanism that prevents us from generating the same oblique plane triangulation multiple times. Doing so is not only inefficient, but also urges us to check whether a newly discovered oblique plane triangulation is not isomorphic to any, already found graph. Fortunately, there is a very simple way to fix both of these issues at the same time. The algorithm takes 1 parameter Δ in the beginning, which is the upper bound of degrees in the target oblique plane triangulation. Therefore all results G_i of the algorithm with such parameter will have a property

$$\forall v \in G_i : deg(v) \leq \Delta \;\;\&\;\; u \in G_i, deg(u) = \Delta$$

As described in the chapter 3 the recursive call starts on a first triangle, splitting the surface into two faces, of which one is open. In order to find all oblique plane triangulations for parameter Δ, we seemingly have to perform the recursive call on triangles of all types admissible in an oblique plane triangulation. However, such attitude would lead to graphs not necessarily complying to above mentioned restrictions for a set Δ.

Lemma 4.2 *Let Δ be an integer greater than 3. Then the set $R = \{G \mid \forall v \in G, deg(v) \leq \Delta \;\;\&\;\; \exists u(deg(u) = \Delta \;\;\&\;\; u \in G)\}$ can be generated, by initiating a recursive call on all plane connected graphs, consisting of vertices a, b, c where $\lambda(c) = \Delta$.*

Proof:
Despite the complexity of Lemma, its meaning is in a sense quite simple. It claims two things.

- If the former triangle consisted of vertices a, b, c where $\lambda(a) \neq d, \lambda(b) \neq d$ and $\lambda(c) \neq \Delta$, then there is no guarantee that a the discovered graph will contain a vertex with a degree of Δ.
- If the former triangle contains a vertex with a degree of Δ, then the discovered graph is guaranteed to contain a vertex with a degree of Δ. □

The point of this Lemma was to clarify what is necessary to produce all oblique plane triangulations with a parameter d. Now we need to introduce a principle that will prevent us from discovering the same graph multiple times. Assume that we performed the recursive call on only one initial triangle a, b, c, where $\lambda(c) = \Delta$. Then the result set of graphs would consist of all graphs that contain the face $\langle\lambda(a), \lambda(b), \lambda(c)\rangle$. Firstly, we need to prove that such set would contain only mutually non-isomorphic graphs.

Lemma 4.3 *Let H_0 be a plane connected graph consisting of three vertices a, b, c, where $a \neq b \neq c \neq a$, and two faces, of which one is closed. Then all graphs that were created from this graph by our algorithm are going to be mutually isomorphic.*

Proof:
Assume all children H_1 of H_0. The only admissible operation is add. Without loss to generality, we create $\Delta - 2$ children, where each is non-isomorphic to each other as the closed face $\langle deg(a), deg(b), deg(c)\rangle$ is in H_1 incident to an edge ab which is incident to a closed face $\langle deg(a), deg(b), deg(d)\rangle$, where $deg(d)$ is different in each child H_1.
In general, assume that all graphs H_i are mutually non-isomorphic, then all graphs H_{i+1} will be non-isomorphic either, as each child of a fixed H_i will be non-isomorphic to all other children of H_i for the very same reason as for H_0. □

Now that we have proven, that starting with a fixed initial triangle yields mutually non-isomorphic graphs, we need a way to to assert that starting with all admissible initial triangles will together yield a set of non-isomorphic oblique plane triangulation.

Lemma 4.4 *Let S be a fixed sequence of all admissible faces for a fixed Δ. Let s_i be an admissible face of type $\langle a, b, c\rangle$ from S, where $a = \Delta$ and $b \neq c$. Then if we perform recursive call on the initial triangle containing the closed face s_i, allowing the algorithm to only use faces s_k, where $k > i$, we will get a set of non-isomorphic oblique plane triangulations as a result.*

As we know, performing the recursive call on an initial triangle, which contains a closed face of type a with all admissible faces available will result in receiving all oblique plane triangulations containing the face of type a. Assume then that the recursive call on an initial triangle containing a face s_i will return an oblique plane triangulation containing a face of type s_j, where $j < i$. That would mean that such graph was not found by the call on s_j, which is in conflict with the fact that a the recursive call on initial triangle with face s_j would return all oblique plane triangulations containing face s_j. □

On closer inspection, this approach avoids starting with faces where at least two vertices are of the same type. Since the arrangement of S does not matter, such faces might be at the very end of the sequence. That implies that when starting with first face of such type, all admissible faces adjacent to the vertex of degree Δ must be of type $\langle \Delta, x, x \rangle$. That however is not possible as all faces incident to the vertex of degree Δ would have to be of the same type, which is in conflict with H_i, being a realisation of an oblique plane triangulation.

4.4 Variation of the algorithm

All the algorithm described so far was good for one use. Finding all oblique plane triangulations with restrictions of Δ. However, we have also set a goal to find an oblique plane triangulation with as high Δ as possible. The algorithm described so far does not fulfill this goal sufficiently. Therefore, we need some kind of heuristics to shorten our way to finding an oblique plane triangulation for bigger Δ.

4.4.1 Restricting admissible vertex degrees

As mentioned before, the size of the tree grows more than exponentially based of the value of Δ. However it is not Δ that affects the size of the tree. It is foremost the amount of faces that we can choose from. Until now, the amount of faces was determined by Δ, that is why we based our complexity estimates on Δ. The point of this modification is forbidding the algorithm to use certain degrees or restricting the amount of vertices of other degrees.

Lemma 4.5 *Let $\gamma(\Delta)$ be complexity function of algorithm based on Δ. Then $\gamma(\Delta)$, where we forbid using j distinct vertex degrees $\in O(\gamma(\Delta - j))$.*

Proof:
As we have learned the complexity is based on the amount of admissible faces. The amount of faces a in a graph with maximum degree Δ with j forbidden vertex degrees can be estimated using simple combinatorics.

$$a = \frac{(\Delta - j)^3}{3!}$$

.

Using the same principle we can estimate the amount of admissible faces for an algorithm restricted to $\Delta - j$.

$$b = \frac{(\Delta - j)^3}{3!}$$

.

As $a = b$, then $\gamma(\Delta)$, where we forbid using j distinct vertex degrees $\in O(\gamma(\Delta - j))$.
$\square$

4.4.2 Restricting the amount of vertices of admissible degrees

From the graphs found by our algorithm, we have observed that they generally contain fewer vertices of degree close to Δ. Therefore, we have decided to apply another modification. Instead of calculating the maximum amount of vertices of degree k as $\lfloor\frac{(\Delta-3)(\Delta+2)}{2k}\rfloor$, we will preset the amount to a value from interval $\langle 0, \lfloor\frac{(\Delta-3)(\Delta+2)}{2k}\rfloor)$. Despite the fact that this modification does not directly reduce the amount of faces, it makes the graph H_i more likely to fail due to restrictions in the choose function. This is desired, as it removes the branches that would search for more rare instances of oblique plane triangulations.

4.4.3 Prioritising certain semi-edges to accommodate the modifications

With the above-mentioned modifications in mind, it is usually quite difficult to complete semi-edges incident to the vertex of degree Δ due to lack of various faces. Having reduced amount of admissible faces and amounts of vertices of certain degrees, there are usually very few options how to complete these semi-edges so that we are left with a potential realization of an oblique plane triangulation. Had we not done any measures, it would frequently happen that somewhere deeply in the recursion the algorithm would find out that it cannot complete a semi-edge incident to the vertex of degree Δ. This event is so frequent that we need to prevent it by primarily completing the semi-edges incident to the vertex of degree Δ and then carrying on with the computation.

The last concept that needs to be mentioned is that we need to change the way we retrieve oblique plane triangulations found by the algorithm. Formerly, we would collect all result into a pool that is returned once we dive out of the recursion. However, in order to retrieve the pool we would have to wait until the the algorithm traverses through the entire search tree which defies the point of these modifications. Hence, we will yield discovered oblique plane triangulations on console or into a file in real time as the algorithm proceeds.

Chapter 5

Implementation

In this Chapter, we are going to address a few implementation details and basic code overview. As of the technology, we have used Java, mainly for it's object oriented design, which was the basic concept of our program. The program was split into several classes whose brief overview will be presented in following sections.

5.1 Class FaceStorage

This class introduces a structure to ease the labor with face types. Its main purpose is to unify all permutations of a face type $\langle x, y, z\rangle$ so that we do not have to sort the items of the triplet x, y, z when referring to a face type. In the algorithm, we will need to check whether a face of a certain type x, y, z is available to be used during recursion. We will also need a function to mark a face type as unavailable or available. The data structure used to achieve this can be most briefly described by a picture.

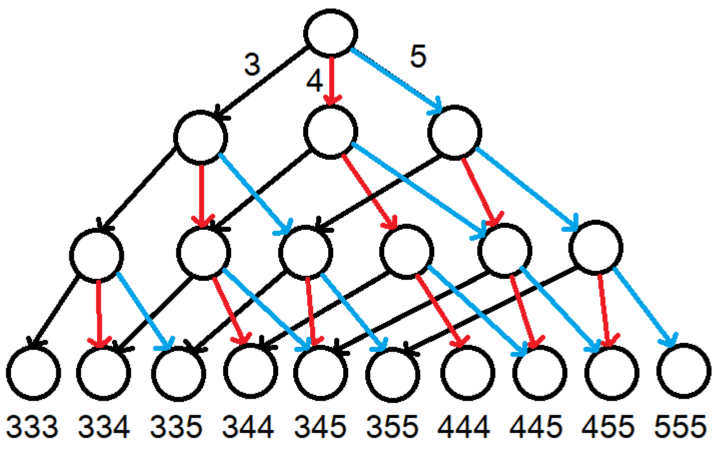


Figure 5.1: This figure shows faceStorage for $\Delta = 5$, where each color represents a value of element in triplet x, y, z

The leaves of this tree-like structure contain *booleans* that represent the availability of that specific face.

5.2 Class Edge

This class represents edge types, more specifically all distinct pairs x, y, where $3 \geq x, y \leq \Delta$. It also contains a list of all open edges of x, y in H_i. The most important attribute however, is the number of available closed faces that can be incident to the edge of this type.

5.3 Class OpenEdge

This class represents one open edge in H_i. It has attributes *start vertex, end vertex* and *face* to which it is incident.

5.4 Class EdgeStorage

Similarly as the class FaceStorage, this class contains all edge types x, y admissible for Δ. The difference is that it has one less layer compared to the FaceStorage and values in the leaves are not booleans but instances of the class Edge.

5.5 Class MutablePriorityQueue

This class is a modification of traditional heap. The difference is that in MutablePriorityQueue, the priority of elements can change after they have been added in the heap. This data structure is necessary, as in order to minimise the width of the search tree, we need to find the edge with least available closed faces that can be incident to it. Therefore, heap-like structure is tailored to this purpose. However we need the aforementioned option to change position of edges in the the heap when their priority changes. We know that that the priority of an edge changed when we do an operation involving an open edge of this type. When such event occurs, we find the instance of that specific type in the EdgeStorage and as each edge remembers its position in the MutablePriorityQueue, we can easily adjust its position.
It is implemented in array and it offers following functionality.

- *Add edge* - adds an Edge to the queue when the first OpenEdge of this type occurs in H_i.

- *Remove edge* - removes and Edge from the queue when the last OpenEdge is removed from H_i.
- *Bubble up* - changes the position of an Edge when an OpenEdge of this type is involved in an operation

5.6 Class Node

This class represents a vertex. Each instance of Node remembers its *label* and a cyclical order of vertices that are adjacent to it, called *neighbours*. *Neighbours* is represented as an array, where each element is either a reference to other Node or *null*, which represents a semi-edge. Node also remembers all open faces it is incident to. It remembers the start position and the end position of each incident face. Therefore, all semi-edges at positions between the start position and end position of each incident face belong to that open face.

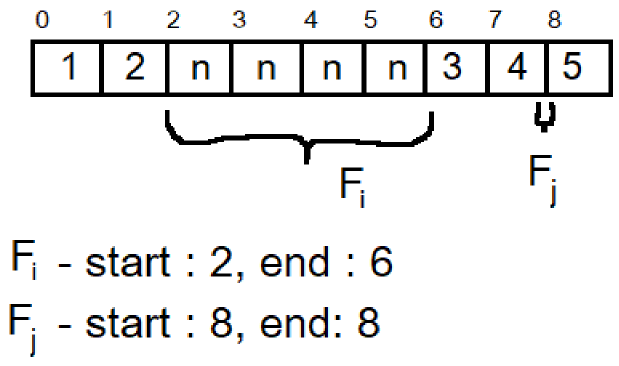


Figure 5.2: Figure represents situation in Node v that is incident to open faces F_i, F_j. Node has four semi-edges in F_i and zero semi-edges in F_j.

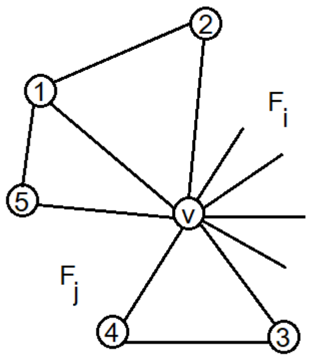


Figure 5.3: Situation in Figure 5.2 corresponds to the situation in this Figure.

Additionally, class Node also contains methods for operations described in Chapter 3. These methods strictly work with attributes of class Node.

5.7 Class Face

This class represent an open face. Each instance of Face F_i remembers cyclical order of vertices that are incident to it. It remembers them in a custom linked list. It also contains methods that transform it during a graph operation.

5.8 Class GraphProcedures

This class contains strictly methods for graph operations described in Chapter 3. According to the parameters described in that Chapter, it performs following actions for each operation.

- Performs respective methods on instances of Face and Node involved in this operation.
- Adds or removes involved instances of OpenEdge into the list of their relative Edge.
- Changes positions of involved instances of Edge in MutablePriorityQueue.
- Checks whether an involved vertex does not have zero semi-edges in an open face, if it does then saves the vertex into a list of "zero semi-edge vertices" in that respective face.
- Checks whether no pair of adjacent vertices involved in the operation has one semi-edge in an open face each. If it does then saves it similarly as in the previous case.

5.9 Class Graph

This class nests all the vertices, faces and other utilities necessary to manage the process of algorithm described in Chapter 3. It also features the structure and methods described in that very same Chapter. It might be referred to as the backbone of the algorithm.

Chapter 6

Results

As mentioned many times before, we have set two goals. The first one was to find all oblique plane triangulations for as high Δ as possible. The second one was to find an oblique plane triangulation containing at least one vertex of degree Δ for Δ as high as possible.

6.1 Finding all oblique plane triangulations restricted by maximum degree

The first achievement was that we confirmed that there are no oblique plane triangulation for $\Delta < 8$. This belief was based on theory that we are not going to cover in this work.
Secondly, we have discovered that there are eleven non-isomorphic oblique plane triangulations for $\Delta = 8$. The computation for $\Delta = 8$ took about 8 seconds. The concept of the algorithm has changed several times in the process in order to make the algorithm more effective for larger Δ. In all iterations we have discovered the same eleven oblique plane triangulations. We are providing all these graphs as an appendix to this paper.
Lastly, we have not been able to find all oblique plane triangulations for any $\Delta > 8$. The reasons for that are apparent from the Chapter 4. We have tried performing the recursive call on an initial triangles containing vertices of degrees $3, 4, 9$ omitting operation "connect". The omission of the operation "connect" was intentional as a time saving precaution. Despite that, we were able to discover about 1400 non-isomorphic oblique plane triangulation in about four hours of computation. This was not very scientific approach as the set of graphs we have found can be hardly described as a mathematical set, therefore we list it as a matter of interesting fact. The best that we can claim that this is a subset of all oblique plane triangulation for $\Delta = 9$, containing a face of type $\langle 3, 4, 9\rangle$.

6.2 Finding oblique plane triangulations with high vertex degrees

We have been able to find oblique plane triangulations for all $8 \leq \Delta \leq 26$ at the point of finishing this paper. We have done so using the modified version of the algorithm described in section 4.4. We are providing all these graphs as an appendix to this paper. Additionally we are providing a following table. The content of this table covers the time necessary to find the first oblique plane triangulation for given Δ with various restrictions. We will be restricting the amount of vertices of degree higher than six. The format $k \times n$ represents that there can be at most k vertices of degree n. If $n > 6$ and n is not covered in the column restrictions, that stands for $0 \times n$.

Δ	restrictions	time	Δ	restrictions	time
10	$1 \times 7, 1 \times 8$	$< 1s$	18	$2 \times 7, 2 \times 8, 1 \times 9$	$180s$
10	$2 \times 7, 2 \times 8$	$< 1s$	18	$3 \times 7, 2 \times 8, 1 \times 9$	$4.5s$
11	$2 \times 7, 2 \times 8$	$< 1s$	18	$3 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$3s$
11	$1 \times 7, 1 \times 8$	3m	18	$4 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$12s$
12	$2 \times 7, 2 \times 8$	$< 1s$	18	$4 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$3s$
13	$2 \times 7, 2 \times 8$	$1s$	18	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$1.8s$
14	$2 \times 7, 2 \times 8$	$3s$	19	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$9s$
14	$3 \times 7, 2 \times 8$	$11.5s$	19	$3 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$< 1s$
14	$3 \times 7, 2 \times 8, 1 \times 9$	$< 1s$	20	$3 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$1.8s$
15	$3 \times 7, 2 \times 8, 1 \times 9$	$1.5s$	20	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$20s$
15	$4 \times 7, 2 \times 8, 1 \times 9$	$< 4s$	20	$4 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$68s$
15	$2 \times 7, 2 \times 8, 1 \times 9$	$< 1s$	20	$4 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$5s$
15	$2 \times 7, 2 \times 8$	$2.5s$	20	$3 \times 7, 2 \times 8, 1 \times 9$	$> 200s$
16	$2 \times 7, 2 \times 8$	$> 60s$	21	$3 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$3.5s$
16	$3 \times 7, 2 \times 8$	$> 60s$	21	$4 \times 7, 2 \times 8, 1 \times 9, 1 \times 10$	$19s$
16	$3 \times 7, 2 \times 8, 1 \times 9$	$3s$	21	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10$	$> 200s$
16	$2 \times 7, 2 \times 8, 1 \times 9$	$1.5s$	21	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10, 1 \times 11$	$< 1s$
16	$2 \times 7, 1 \times 8, 1 \times 9$	$3s$	22	$3 \times 7, 2 \times 8, 2 \times 9, 1 \times 10, 1 \times 11$	$136s$
17	$2 \times 7, 1 \times 8, 1 \times 9$	$9s$	22	$3 \times 7, 2 \times 8, 1 \times 9, 1 \times 10, 1 \times 11$	$105s$
17	$2 \times 7, 2 \times 8, 1 \times 9$	$1s$	22	$4 \times 7, 2 \times 8, 2 \times 9, 1 \times 10, 1 \times 11$	$481s$

Table 6.1: For various combinations of values of maximum degree and restrictions on numbers of medium-degree vertices, the time to produce a first graph is shown.

Chapter 7

Conclusion and perspectives

When we started the work, we set out a goal to find at least one oblique plane triangulation using our algorithm. At the time of writing this paper, we have found overwhelming number of non-isomorphic oblique plane triangulations which can be considered as a great success. The state of the algorithm is very good and we do not think it can be improved drastically, however some tweaks might be done in order to perfect its performance. If we had more time at our disposal, I believe it is in our computational power to find all oblique plane triangulations with maximum degree of 9. On the similar note, we might be able to find oblique plane triangulations with higher maximum degree of an vertex. Overall, we are satisfied with the results and consider them as an interesting contribution.

Bibliography

[1] František Kardoš and Jozef Miškuf. Maximum vertex and face degree of oblique graphs. *Discrete Mathematics*, 309(15):4942 – 4948, 2009. Cycles and Colourings.

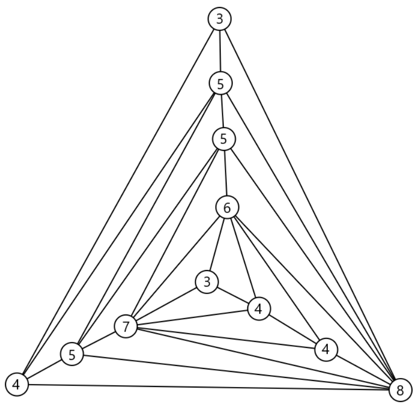
[2] Jens Schreyer. Oblique graphs. Dizertačná práca na získanie akademického titulu Doctor rerum naturalium, 2005.

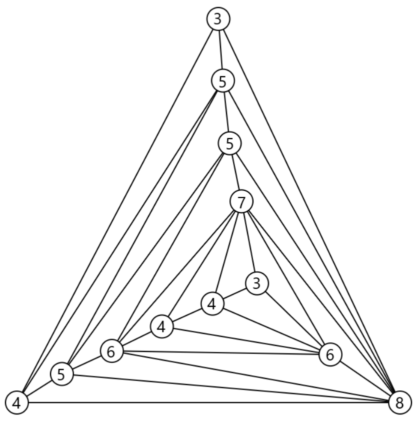
[3] M. Voigt and H. Walther. Polyhedral graphs with restricted number of faces of the same type. *Discrete Mathematics*, 244(1):473 – 478, 2002. Algebraic and Topological Methods in Graph Theory.

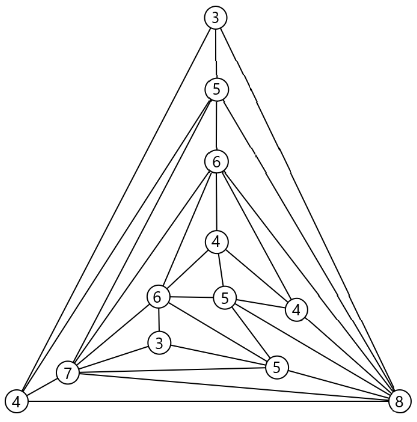
[4] Hansjoachim Walther. Polyhedral graphs with extreme numbers of types of faces. *Discrete Applied Mathematics*, 120(1):263 – 274, 2002. Special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

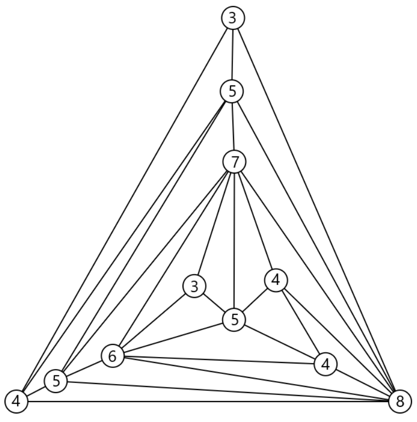
Appendix A

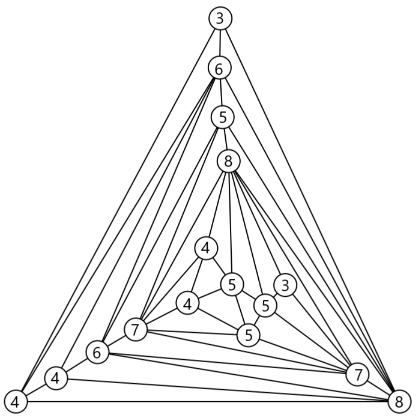
The eleven oblique plane triangulations with maximum degree 8.

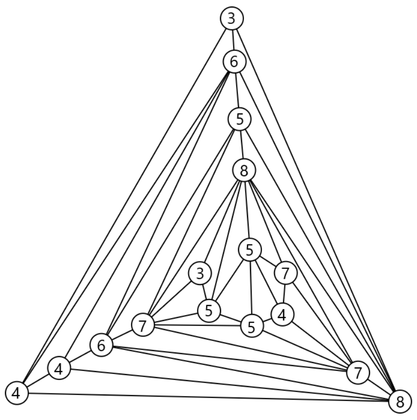


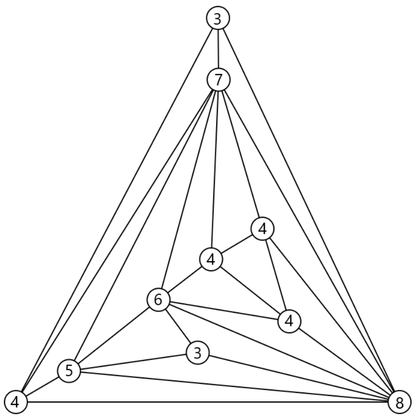


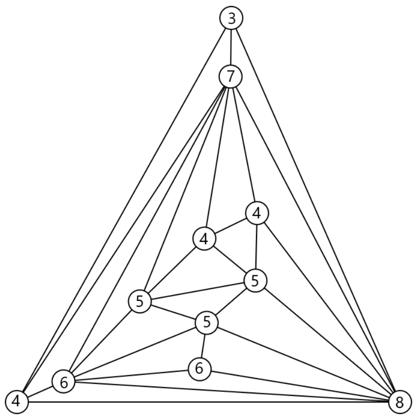


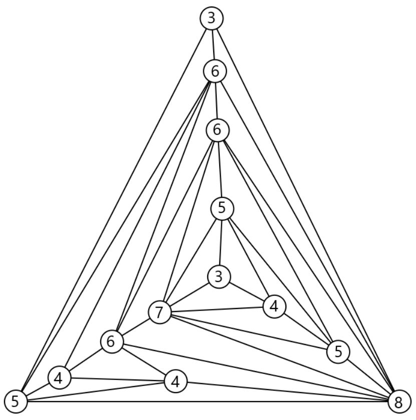


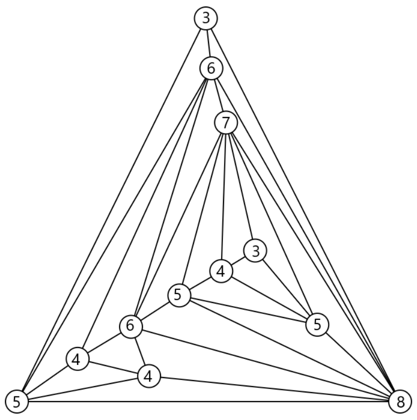

3
6
5
8
4
5
3
4
5
7
5
6
7
4
4
8

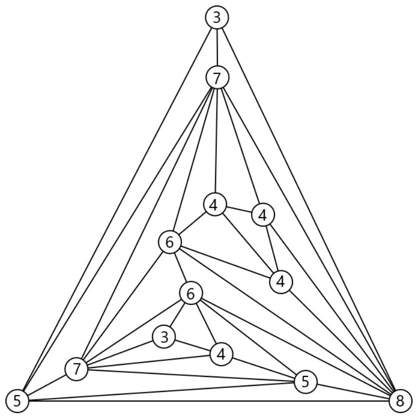

3
6
5
8
5
3
7
5
4
7
5
6
7
4
4
8


3
7
4
4
6
4
3
5
4
8


3
7
4
4
5
5
5
6
6
4
8


3
6
6
5
3
7
4
6
5
4
4
5
8

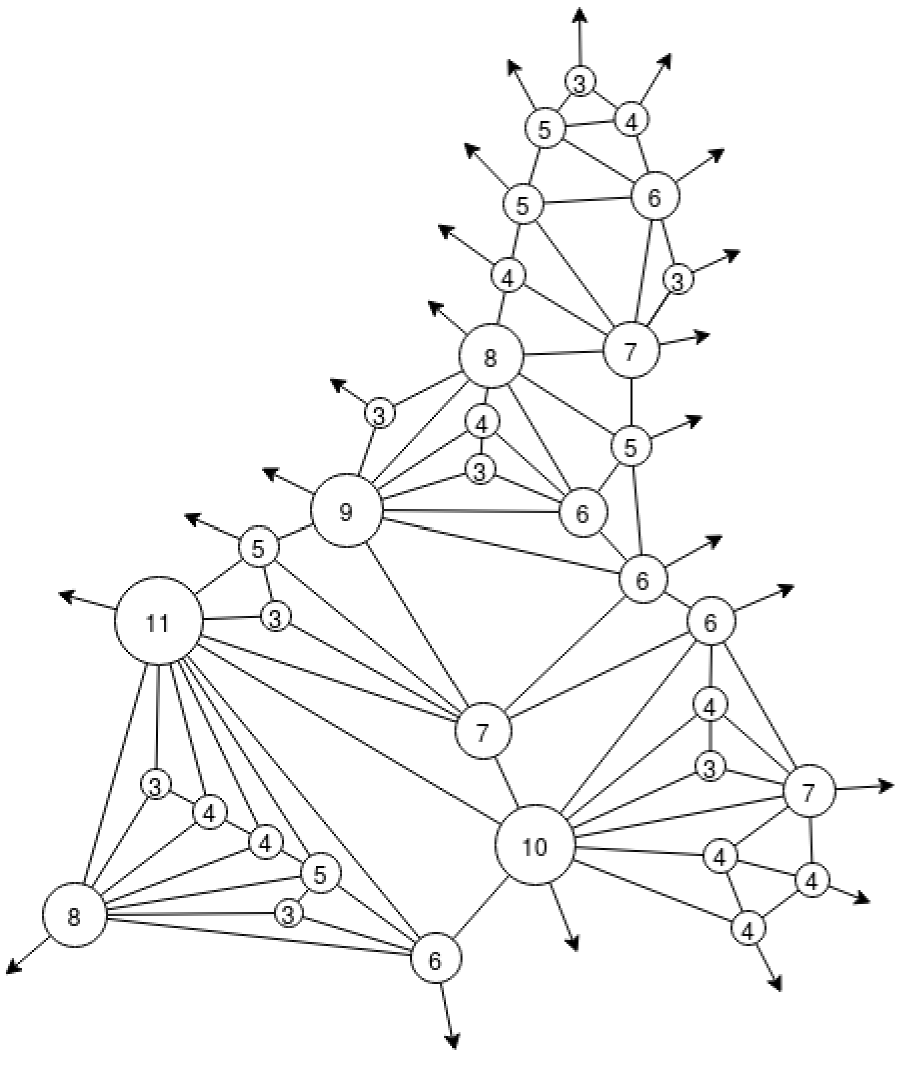

3
6
7
3
4
5
6
5
4
4
5
8

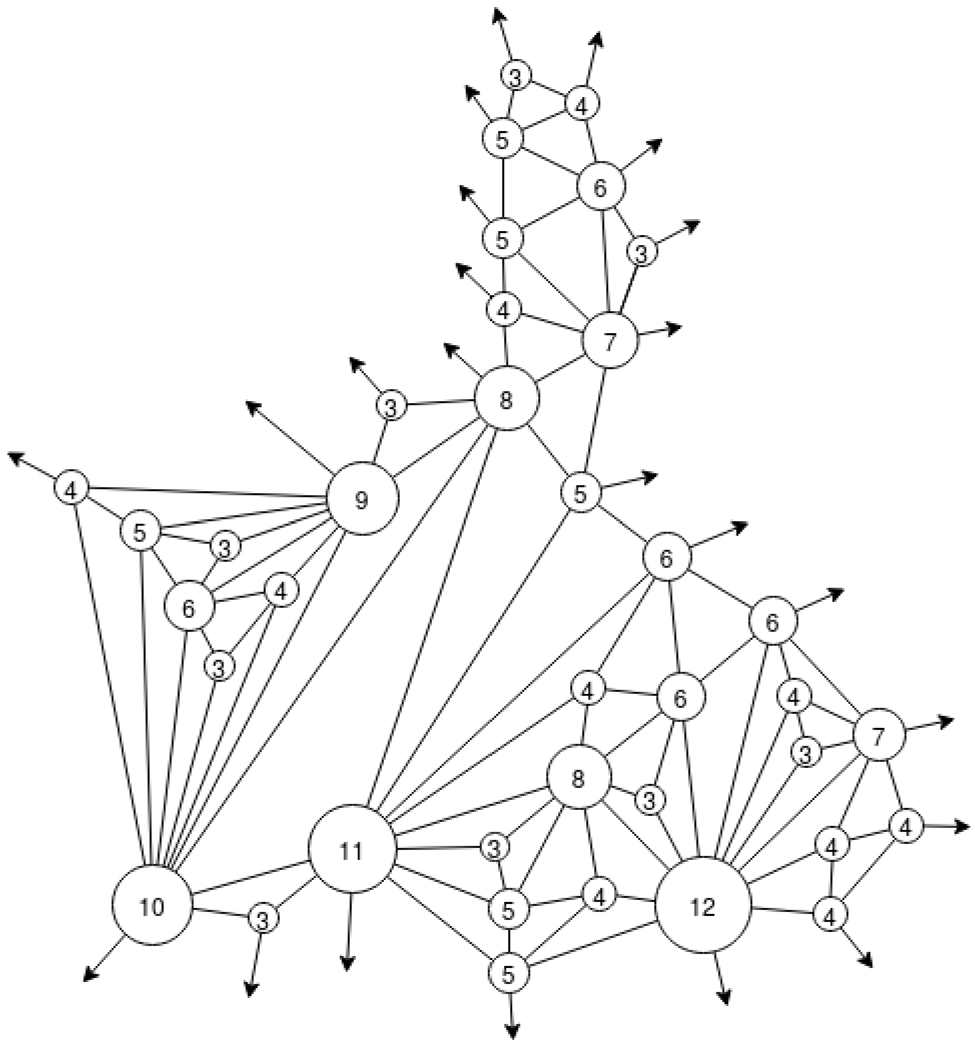

3
7
4
4
6
4
6
3
4
7
5
5
8

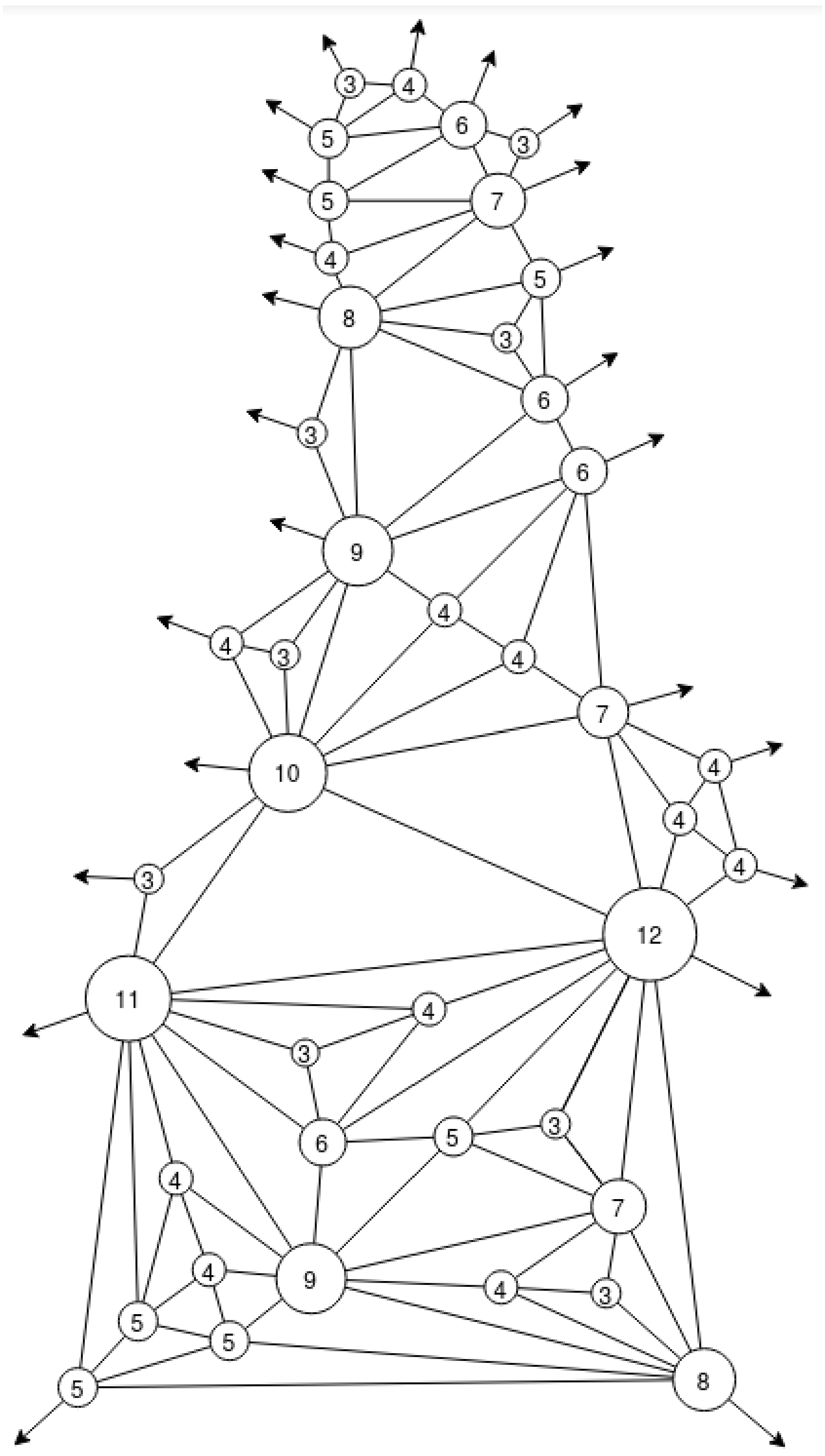
Appendix B

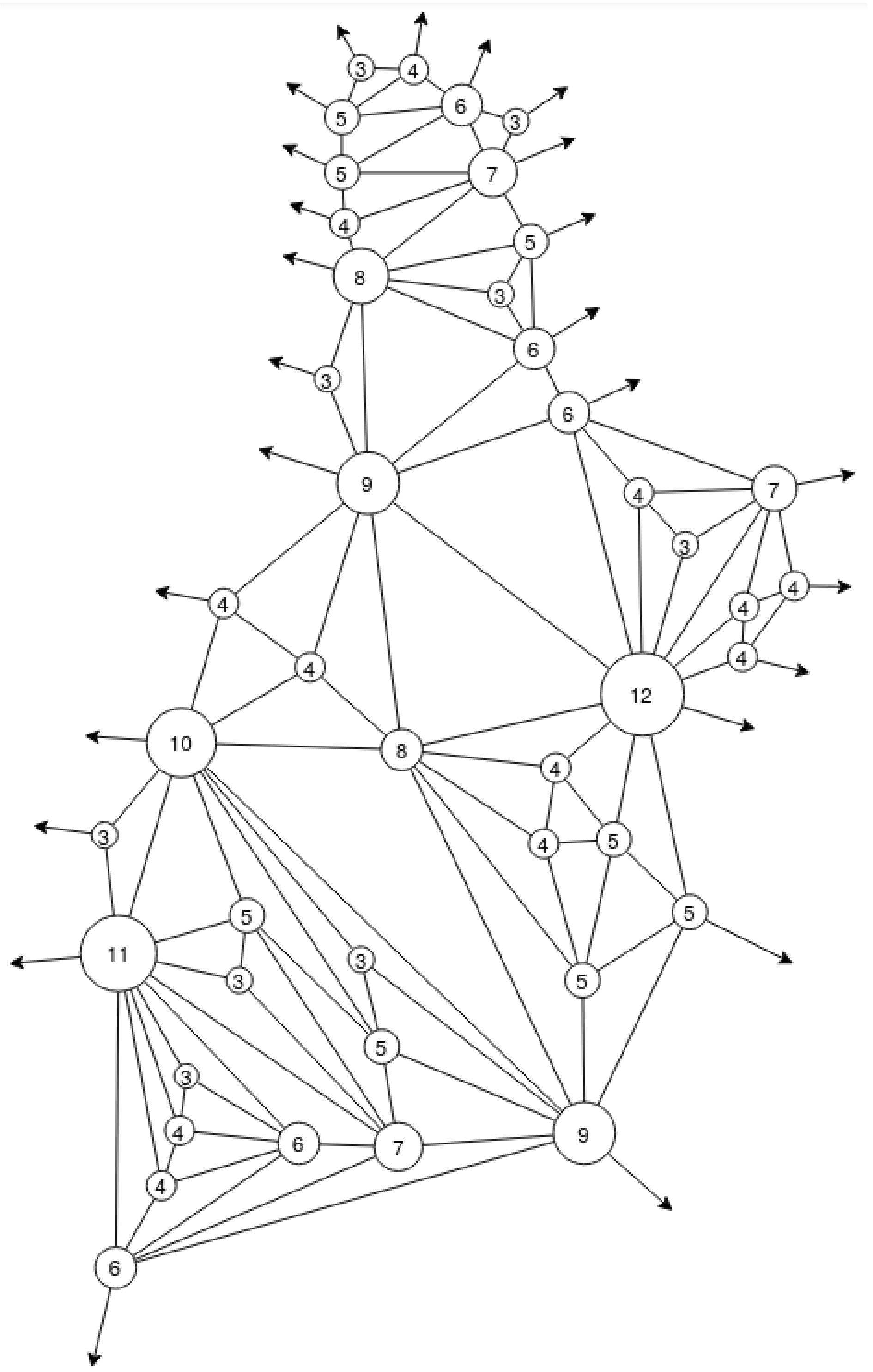
Representatives for each set of oblique plane triangulations with maximum degrees 22-26.

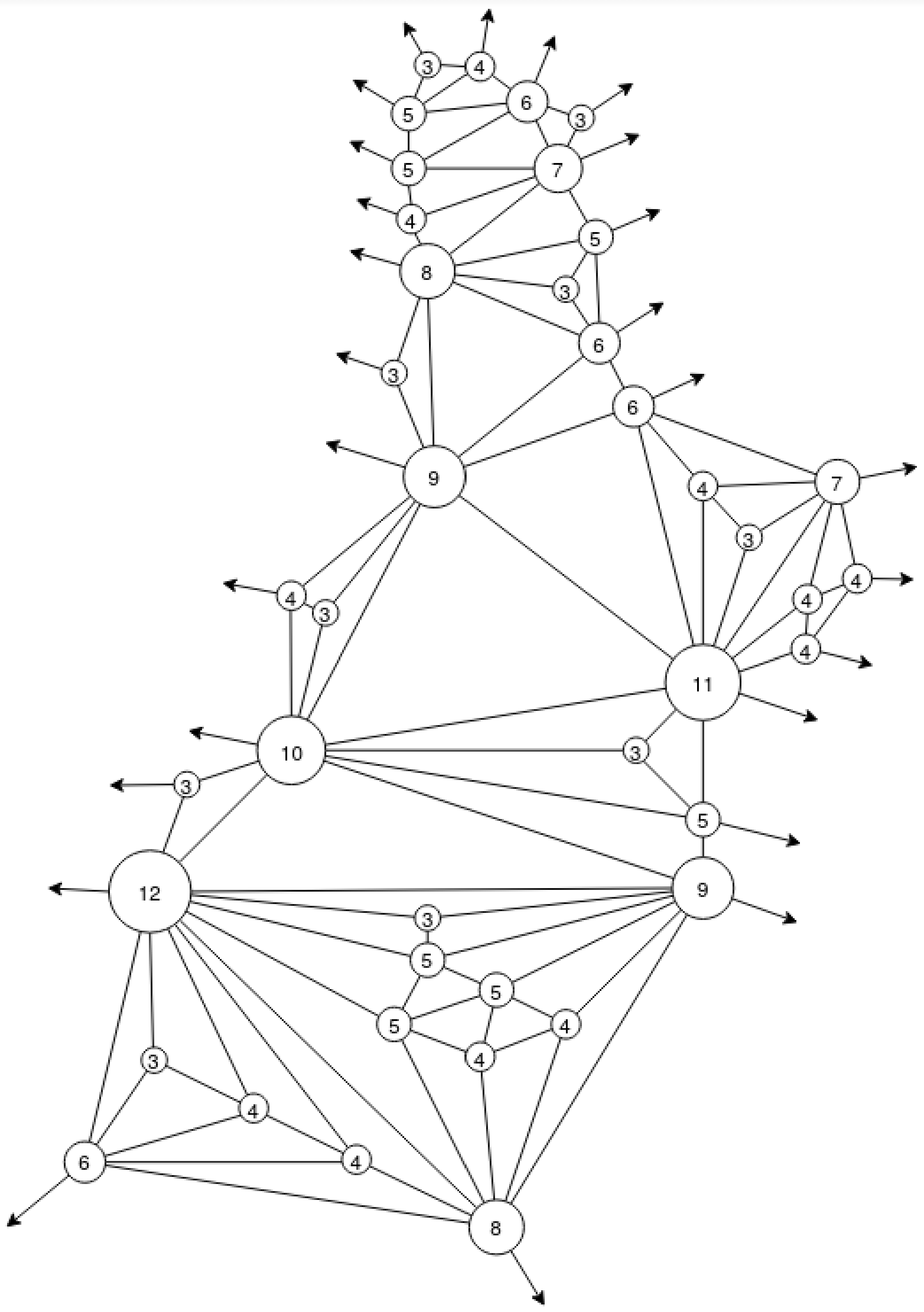
The edges represented by arrow are incident to the same vertex with degree 22-26 for each graph respectively.




3
4
5
6
5
3
4
7
8
3
9
4
5
3
6
4
3
5
6
6
4
6
4
7
3
8
3
11
3
4
4
10
3
5
4
12
4
5


3
4
5
6
3
5
7
4
5
8
3
6
3
6
9
4
3
4
4
7
10
4
4
3
4
12
11
4
3
6
5
3
4
7
4
9
4
3
5
5
5
8


3
4
5
6
3
5
7
4
8
5
3
6
3
6
9
4
7
3
4
4
4
4
4
12
10
8
4
3
4
5
5
11
5
3
3
5
5
3
4
6
7
9
4
6


3
4
5
6
3
5
7
4
5
8
3
6
3
6
9
7
4
3
4
4
4
4
3
11
10
3
3
5
12
9
3
5
5
5
4
4
3
4
4
6
8