Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: Algorithmic Computer Science **(INA)**

ENGINEER'S THESIS

Algorithms for the minimum genus of a graph

Jarosław Grzegorz Socha

Supervisor
PhD. Małgorzata Sulkowska

Keywords: graph, minimum genus, ILP, QUBO, quantum annealing

WROCŁAW 2024

Abstract

The goal of this thesis was to investigate the application of quantum annealers to solve the minimum genus of a graph problem. Various *Integer Linear Programming* (ILP) and *Satisfiability* (SAT) formulations of the problem were explored and translated into the *Quadratic Unconstrained Binary Optimization* (QUBO) model, which is well-suited for quantum annealers. A theoretical analysis of the models' sizes was conducted, along with empirical testing on representative graphs and the *Rome* graph dataset. The findings suggest that different models may perform better, depending on the graph's maximum vertex degree and density. However, for most of the graphs studied, the problem proved unsolvable with current quantum annealer capabilities, though this may change as the technology is being rapidly developed.

Streszczenie

Celem pracy było zbadanie zastosowania technologii wyżarzaczy kwantowych do problemu wyznaczania minimalnego genusu grafu. Różne modele *Binarnego Programowania Liniowego* (ILP) i *Problemu Spełnialności* (SAT) zostały opisane i przetłumaczone na instancje *Programowania Kwadratowego Bez Ograniczeń* (QUBO), które są obsługiwane przez wyżarzacze kwantowe. Przeprowadzona została analiza teoretyczna rozmiarów formulacji, a także doświadczalnie zbadano wielkości modeli dla kilku grafów reprezentatywnych i dla grafów ze zbioru *Rome*. Z testów wynika, że w zależności od maksymalnego stopnia wierzchołka w grafie, jak i gęstości grafu, różne modele mogą być bardziej odpowiednie, aczkolwiek problem genusu dla większości badanych grafów okazał się nierozwiązywalny na obecnie rozwijanych wyżarzaczach kwantowych, choć z uwagi na szybki rozwój tej technologii, sytuacja ta może ulec zmianie w przyszłości.

Contents

1. Introduction

Given a problem in graph theory, it is beneficial to know something about the examined graph, such as its maximum degree or connectivity, as knowing some of its properties may simplify the problem or reduce the calculation time and resources. The minimum genus of a graph is one such property. Obtaining a high bound for the genus of a graph can speed up some algorithms, especially if an embedding with such genus is also given [3], and the best high bound is the exact solution. Additionally, many problems in mathematics require knowing the minimum genus of a graph, which can be tedious when it comes to a formal proof, and easy with an algorithmic approach [6, 23]. To solve the problem in question, a novel type of computer might be used, called a quantum annealer. Such a device utilizes quantum processes to solve optimization problems, and might prove to be a good alternative to classical solvers in the near future. The goal of this thesis is to convert the known models developed for solving the minimum genus of a graph problem into a form accepted by a quantum annealer. Afterwards, the thesis aims to answer whether the problem in question is well suited for exploration on such a device, by conducting a theoretical and empirical size analysis of the different models.

First, theoretical concepts of graph theory, minimum genus problem solving and *Linear Programming* will be introduced. Then, the different models for solving the problem will be explored. After that, the model accepted by a quantum annealer, the *Quadratic Unconstrained Binary Optimization* (QUBO) model, will be shown, along with the process of converting an *Integer Linear Programming* (ILP) instance into it. Next, QUBO formulations of the known models for the minimum genus of a graph problem will be derived, with an analysis of their sizes conducted. Afterwards, the exact method of solving the problem will be explained, with the necessary preprocessing steps and the specificity of solving the problem on a quantum annealer. Lastly, some experiments will be conducted to see, how current quantum computers are suited for solving the given problem, on benchmark graphs.

2. Theoretical concepts

2.1. Graph

A graph G is a pair $G = (V, E)$, where V is a set of nodes, and E is a multiset of edges, which are subsets of V of size 1 or 2. It can be either undirected, or directed, where each edge of size 2 becomes an ordered pair instead of being a subset, while edges of size 1 stay. When it comes to the problem in question, only undirected ones will be considered, as edge direction doesn't impact the minimal genus problem. A graph in a general sense can have many edges between two nodes (called multi-edges) or edges starting and ending in the same vertex (called loops), due to E being a multiset and having elements of size 1. However, none of those types of edges affect the problem, so whenever the input graph G contains any of those, we can remove the loops and merge the multi-edges, creating a new set of edges $\overline{E}$ which is a subset of E with elements of size 2 only. A new considered graph $\overline{G} = (V, \overline{E})$, where $\overline{E}$ is a set of 2-sized subsets of V, is called a simple graph.

For a simple graph $G = (V, E)$, we define the neighbourhood $N(v)$ of a vertex $v \in V$ as a set of vertices connected to v, $N(v) = \{u \in V : \{u, v\} \in E\}$. We also define the degree of a vertex $v \in V$ as the number of its neighbours $deg(v) = |N(v)|$. The maximum degree of a graph $\Delta(G)$ is the maximum degree of its vertices.

A subgraph of a graph G is the graph G with any number of its vertices, along with their incident edges, removed and any number of additional edges removed.

Edge contraction on a graph $G = (V, E)$ is an operation, which takes an edge $\{v, w\}$, deletes it from the graph and merges both it's endpoints v and w into one vertex $\overline{vw}$, creating a new graph $\overline{G} = (\overline{V}, \overline{E})$. This operation preserves the original vertices' connections to other vertices, so $N(\overline{vw}) = \{u : \{u, v\} \in E \ \vee \ \{u, w\} \in E\}$. Note worthily, $max\,(deg(v), deg(w)) \leq deg(\overline{vw}) \leq deg(v) + deg(w)$.

A minor of a graph G is a subgraph of G, with any number of its edges contracted.

2.2. Genus

Formally, genus is a property of a surface. In this paper, only the genus of an orientable surface will be discussed, which intuitively is a surface that splits three-dimensional space into two regions, an inside and an outside region. Examples of such surfaces are a sphere and a toroid. A counterexample to those would be a Möbius strip, which is a non-orientable surface.

The genus of an orientable surface is the natural number corresponding to the maximum number of closed cuts, which can be made without splitting the surface into two surfaces, where a closed cut is a cut along a continuous line starting and ending at a common point.

A function between two surfaces is a homeomorphism if it is bijective and its inverse along with itself are both continuous. Two orientable surfaces are homeomorphic if and only if they have the same genus. The surface of the lowest genus, which is 0, is homeomorphic to a sphere.

In simpler terms, we can think of the genus twofold: either as the number of holes in a sphere, or as the number of handles added to a sphere [see Figure 2.1].

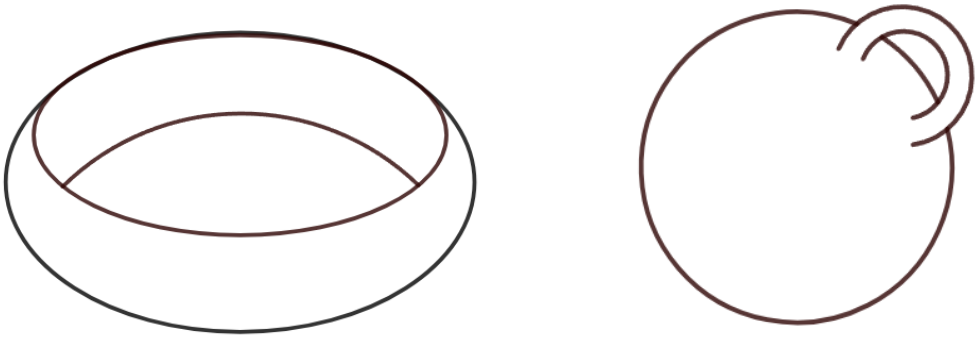

Figure 2.1: Two depictions of surfaces homeomorphic to a torus. The first one can be thought of as a sphere with one hole, and the other as a sphere with one handle

2.3. Graph embedding

A graph drawing on a given surface is a function between graph elements and points of the surface, where vertices are mapped to distinct points and edges are assigned to continuous curves between incident points. A graph embedding is a drawing of a graph on a surface, such that two edges intersect only at vertices, and only if they are incident to those vertices. Such drawing splits a surface into faces, regions of the surface bounded by graph edges and vertices. Every graph can be drawn on a surface of an arbitrarily high genus, although it is beneficial to choose a surface, for which every hole or handle is used, as such a surface has lower genus, which leads to less computation and faster execution for many algorithms [3]. More formally, it would be a surface for which every face of the graph embedded on it is homeomorphic to an open disc, with an open disc being the set of points inside a circle drawn on a surface, without the circle itself. Such an embedding is called a 2-cell embedding [2]. However, simply finding any 2-cell embedding of a graph doesn't always result in a drawing of a graph with the lowest possible genus, as two 2-cell embeddings of the same graph can have different genera (see Figure 2.2). This necessitates differentiating between embeddings, to accomplish the goal of finding one, for which the genus is the lowest.

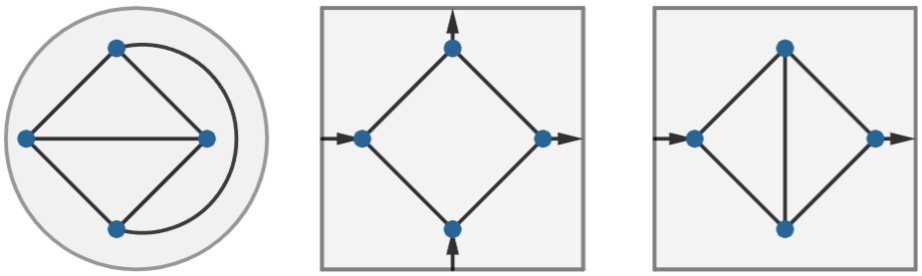

Figure 2.2: Different embeddings of the same graph G. The first one is a 2-cell embedding of G on a sphere, which is the minimal surface on which G can be embedded. The second one is also a 2-cell embedding of G, but on a torus, where the surrounding rectangle is a standard representation of a torus. Whenever an edge enters a side of the rectangle, it exits from an opposite point on the other side of the rectangle, which is signified by the arrows. The third figure represents an embedding of G on a torus, that is not a 2-cell embedding, as the outer face is homeomorphic to a disc with a hole in it

2.4. Rotation systems

A rotation system is a way to model an embedding of a graph $G = (V, E)$. One such way is to use a following system: $\pi = \{\pi_v : v \in V\}$, where π_v is a cyclic permutation (a permutation consisting of a single cycle) of edges connected to v, which specifies how the neighbours of v are oriented around the vertex v [4]. One could argue that only defining such a rotation system isn't enough, because different graph embeddings can have the same π [see Figure 2.3] [4]. However, as long as two 2-cell embeddings follow the same rotation system, the genera of the surfaces they are embedded upon are the same [15, Theorem 3.2.3] [4, Lemma 1.10]]. As such, we can define the genus of a rotation system as the genus of a surface of any 2-cell embedding following that system.

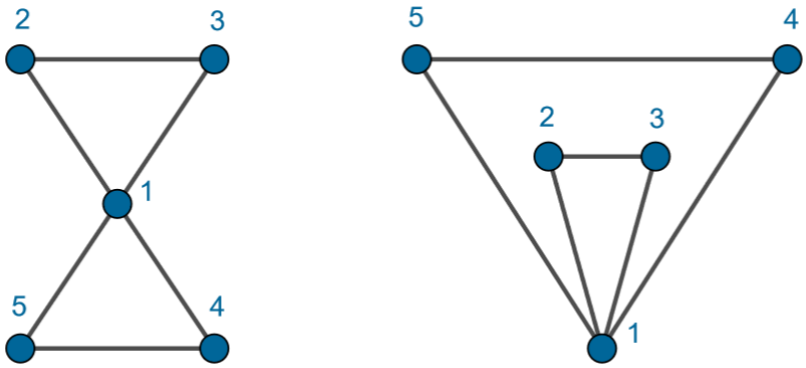

Figure 2.3: Two different embeddings of a graph with the same π. As an example, both drawings have $\pi_1 = \left(\begin{smallmatrix} 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 2 \end{smallmatrix}\right)$

2.5. Minimum genus of a graph

Minimum genus of a graph corresponds to the minimal genus of a surface for which there exists a 2-cell embedding of the graph. For example, a graph K_4 (a clique consisting of 4 vertices, all connected to each other) is planar, so its minimum genus is 0. On the other hand, a graph K_5 isn't planar, because it cannot be embedded on a sphere, but it can be embedded on a torus, so its minimum genus is 1.

Every 2-cell embedding on an orientable surface of a graph $G = (V, E)$ with $|F|$ faces, satisfies the Euler's formula

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

where g is the genus of the surface [33] [15, Theorem 3.3.3].

2.6. Number of faces

For the algorithms in this thesis, it is sometimes necessary to calculate the maximum number of faces for a given embedding of a graph $G = (V, E)$, as it speeds up some calculations. The graphs considered here will be non-planar, as dedicated algorithms exist for determining graph planarity. Then $|V| > 3$ and $|E| > 3$.

For a graph embedding, every edge appears as a boundary of a face exactly two times. Additionally, every face consists of at least three edges. If we were to assume, that every face is the smallest it can be, we get the first theoretical high bound for the number of faces $f_1 = \left\lfloor \frac{2|E|}{3} \right\rfloor$.

Another approach is to use the Euler's formula. As the graphs are non-planar, any embedding of G has genus at least 1. We obtain the highest number of faces when genus is the lowest, so $f_2 = |E| - |V|$.

Other, more complicated but better bounds have been computed and described in detail [6], and even better ones may be found in the future. As such, by $\bar{f}$ we will denote the lowest upper bound for the number of faces for the graph, which can be adapted for any future findings.

3. Finding the minimum genus of a graph

3.1. Genus of a rotation system

The first step in finding the minimum genus of a graph is determining the genus of a single rotation system. To do so, we take any 2-cell embedding which follows a given rotation system. From the relation between the properties of a graph embedding and genus $|V| - |E| + |F| = 2 - 2g$, it can be deduced that finding the number of faces of an embedding is enough to calculate the genus of a rotation system, as the number of vertices and edges is the same for every embedding. The basic idea for this procedure is to use the face tracing algorithm.

3.2. Face tracing algorithm

As an input, the algorithm takes rotation system π of graph $G = (V, E)$ [1]. The idea of the algorithm is to trace the edges in the order that their vertices appear in the permutations. The first step is to take G and change it into a directed graph, where every undirected edge $\{v, w\}$ turns into two directed arcs (v, w) and (w, v). Such operation makes it, so each arc belongs to exactly one face. In that scenario, a face is a sequence of arcs, where after every arc (v, w) comes an arc $(w, \pi_w(v))$, as $\pi_w(v)$ will be the next vertex after w.

Algorithm 3.1: Face tracing algorithm [3]

```
Data: Graph G, rotation system π
Result: number of faces f
1 f ← 0
2 foreach arc (v, w) of G do
3     while (v, w) is unused do
4         mark (v, w) as used
5         (v, w) ← (w, π_w(v))
6     f ← f + 1
```

As throughout the loop edges are dynamically marked as used, the number of faces of a rotation system π can be found in $O(|E|)$.

[1] even if G is not given, then it can be easily deduced from π

3.3. Problem hardness

Although finding the genus of a 2-cell embedding can be done in polynomial time, finding the minimum genus of a graph is harder. For each vertex $v \in V$ of $G = (V, E)$, there are $(deg(v) - 1)!$ different rotations of $N(v)$ around v. A rotation system consists of those different rotations, so there are $\prod_{v \in V} (deg(v) - 1)!$ different rotation systems for a given graph. This is an upper bound for the number of rotation systems that need to be checked, as some rotation systems may produce similar embeddings [25].

While explicit formulas for genera of some graph families have been discovered [32, Appendix A: Graphs with Known Genera] (for example, genus of a complete graph K_n for $n \geq 3$ is $\left\lceil \frac{(n-3)(n-4)}{12} \right\rceil$), for the general case the problem of finding the minimum genus of a graph has been established to be NP-hard [30]. However, it has also been proven to be fixed parameter tractable, where a fixed parameter tractable problem is a problem that can be solved in $O(f(k)n^{O(1)})$ time, with k being a fixed parameter [14] and n characterizing the variable problem size. This is due to the Robertson–Seymour theorem, which states that for every surface, there is a finite list of forbidden minors [27]. It means that for a list L of those minors for a given surface, a graph G is embeddable in the surface only if none of the graphs $H \in L$ are minors of G. For a fixed H, checking if H is a minor of G can be done in $O(n^2)$ [19], with n being the number of vertices of G. Recently, it has been shown that it can be also done in almost linear time $(O_H(n^{1+o(1)}))$ [21]. However, there are a few problems. The first problem comes from the constant factors associated with the algorithms, because although the checking can be done even in almost linear time, the O notation hides constants associated with the H graph, which can be very big, to the point that experimentally the algorithm doesn't behave like a linear one. The second problem arises in the size of the list of forbidden minors for different surfaces. For a sphere ($g = 0$) this list contains two graphs, K_5 and $K_{3,3}$ (see Figure 3.1), which is also known as Wagner's theorem [33]. For orientable surfaces of higher genera, those lists are unknown due to their sizes, as even for the smallest of the surfaces, the torus, the list contains over 17500 minors [23]. Other fixed parameter tractable algorithms have been developed [24], although they also suffer from scalability issues and aren't viable in practice [23], or even have mistakes in them [26].

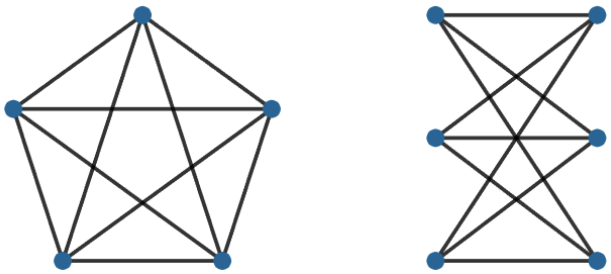

Figure 3.1: Graph K_5 on the left and graph $K_{3,3}$ on the right

4. Linear programming

One of the ways to tackle the hardness of finding the minimal genus of a graph, is by using *Linear Programming*. A special type of linear programming, called *Integer Linear Programming*, is commonly used to solve NP-hard problems, so below the basic linear programming concepts will be explained.

4.1. General model

Linear programming (LP) is an optimization method. It consists of real variables $x_1, \ldots, x_n$, an objective function f, which contains those variables, but only in linear form, and a set of linear inequalities, called constraints. The goal of this method is to find values of $x_1, \ldots, x_n$ that maximize the objective function, without violating the constraints. Below is the general form of a linear program, for a vector c of n objective function parameters, matrix A of size $k \times n$ of constraint parameters and a vector b of k free terms of constraints, for an arbitrary $k \geq 0$

$$\begin{array}{lccccccccccc} max & c[1]\cdot x_1 & + & c[2]\cdot x_2 & + & \ldots & + & c[n]\cdot x_n & & & & \\ & A[1,1]\cdot x_1 & + & A[1,2]\cdot x_2 & + & \ldots & + & A[1,n]\cdot x_n & \leq & b[1] \\ & A[2,1]\cdot x_1 & + & A[2,2]\cdot x_2 & + & \ldots & + & A[2,n]\cdot x_n & \leq & b[2] \\ s.t. & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ & A[k,1]\cdot x_1 & + & A[k,2]\cdot x_2 & + & \ldots & + & A[k,n]\cdot x_n & \leq & b[k], \end{array}$$

where *s.t.* is a short form of *subject to*. This can be written in a compact form

$$max \quad \sum_{i=1}^{n} c[i]\cdot x_i$$

$$s.t \quad \sum_{i=1}^{n} A[1,i]\cdot x_i \leq b[1]$$

$$\vdots$$

$$\sum_{i=1}^{n} A[k,i]\cdot x_i \leq b[k],$$

or in a matrix form, where x is a vector of variables $x_1, \ldots, x_n$ and all the vectors are vertical

$$max \quad c^T x$$

$$s.t. \quad A\,x \leq b.$$

Sometimes, constraints can take the form of equalities instead of inequalities. Those are just a special case of the model above, as any equality in the form of

$$\sum_{i=1}^{n} a_i \cdot x_i = a_{n+1},$$

where variables a are parameters, can be written as two inequalities

$$\sum_{i=1}^{n} a_i \cdot x_i \leq a_{n+1}$$

$$\sum_{i=1}^{n} a_i \cdot x_i \geq a_{n+1}.$$

Equalities are still used in many model descriptions, as they can be more intuitive.

4.2. Integer linear programming

In a traditional linear programming instance, the variables can take values of any real number. In integer linear programming (ILP) a new type of constraint is added for some variables, that restricts them to integers. While linear programming belongs to the P-hard class of problems [13], adding the integer constraint moves the problem into the NP-hard class [20]. There is a subset of integer linear programming called binary linear programming, where variables take binary values. This will be the main model used in this thesis.

5. ILP models

For the minimum genus of a graph problem, many ILP models have been proposed [3, 8]. As the models share a common basis, we will separate parts of them into different modules for ease of use. For some of the models shown below, only Boolean Satisfiability (SAT) formulations are provided in the literature, which are another method of solving NP-hard problems. Those formulations had to be translated into ILP models.

5.1. Base

The basic idea of any model is to simulate the face tracing algorithm numerically, obtaining the equivalent of an embedding with a maximum number of faces.

5.1.1. Prerequisites

It is necessary to mention that the algorithm below works for graphs, where the minimum degree of each vertex is 3. It will also be assumed that the input graph isn't planar. Given a graph $G = (V, E)$, we divide every edge into two directed arcs, one in each direction, just as in the face tracing algorithm. This set of arcs will be denoted by A.

Let $\bar{f}$ be the upper bound for graph's faces mentioned in a previous section (see 2.6). Then $[\bar{f}]$ will be the set of integers from 1 to $\bar{f}$.

5.1.2. Variables

The first set of binary variables is $x_1, \ldots, x_{\bar{f}}$. Each variable x_i is responsible for determining if a face i exists. We will number graph embedding's potential faces with integers from 1 to $\bar{f}$. The variable x_i is 1 only if face i exists, otherwise it is 0.

The second set contains the variables c_a^i for every $i \in [\bar{f}]$ and $a \in A$. A variable c_a^i will be 1 only if arc a belongs to the face i.

5.1.3. Objective function

For every model, the objective function is the same. The goal is to maximize the number of faces

$$max \quad \sum_{i \in [\bar{f}]} x_i. \tag{5.1}$$

5.1.4. Constraints

The constraints below ensure that the values assigned to the variables represent a real embedding of G.

Each face, if it exists, must contain at least three arcs. This prevents the creation of faces consisting of only two, or even zero, arcs. In other words, each face must belong to at least three arcs

$$\sum_{a \in A} c_a^i \geq 3\, x_i \qquad \forall i \in [\bar{f}]. \tag{5.2}$$

Each arc has to belong to exactly one face

$$\sum_{i \in [\bar{f}]} c_a^i = 1 \qquad \forall a \in A. \tag{5.3}$$

For each face a path of arcs containing it must be continuous, so if there are any arcs going into a vertex, the same number must exit. We will denote the set of arcs going into a vertex $v \in V$ by $\delta^+(v)$, and those going out of v as $\delta^-(v)$

$$\sum_{a \in \delta^+(v)} c_a^i = \sum_{a \in \delta^-(v)} c_a^i \qquad \forall i \in [\bar{f}], v \in V. \tag{5.4}$$

The constraints 5.2, 5.3 and 5.4 aren't enough to describe an embedding, but they are the basis on which the following systems lay.

5.2. Describing the rotation system

The base model, while describing basic embedding properties, has to be enriched with information about the rotation system. For it, we first define a rotation system $\pi = \{\pi_v : v \in V\}$ as in a previous chapter.

5.2.1. Successor model

In this approach, we define a variable $p_{u,w}^v$ for each $v \in V$ and for every $u, w \in N(v)$, such that $u \neq w$. The variable is equal to 1 only if $\pi_v(u) = w$.

First, we need to connect the new variables to the base model. It can be done by stating that if $p_{u,w}^v = 1$, then $c_{(u,v)}^i = c_{(v,w)}^i$, which means that if w is the successor of u in the rotation around v, then edges (u, v) and (v, w) must belong to the same face. In the ILP, this condition will be written as two terms

$$\begin{aligned} c_{(u,v)}^i &\geq c_{(v,w)}^i + p_{u,w}^v - 1 \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w, \\ c_{(v,w)}^i &\geq c_{(u,v)}^i + p_{u,w}^v - 1 \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w. \end{aligned} \tag{5.5}$$

Next we need to ensure that variables representing π are unique, so for a given $v \in V$ and $u \in N(v)$ there are no different vertices $v_1, v_2 \in N(v)$, such that $p_{u,v_1}^v = p_{u,v_2}^v = 1$ and also

there are no different vertices $v_1, v_2 \in N(v)$, such that $p^v_{v_1,u} = p^v_{v_2,u} = 1$, which in terms of ILP can be written as

$$\sum_{w \in N(v) \setminus u} p^v_{u,w} = 1 \qquad \forall v \in V, u \in N(v)$$
$$\sum_{w \in N(v) \setminus u} p^v_{w,u} = 1 \qquad \forall v \in V, u \in N(v). \tag{5.6}$$

Lastly, the permutation π must be cyclic. This means, that for $v \in V$, for any non-empty proper subset of neighbours of v $\emptyset \neq U \subsetneq N(v)$, there exists an element $u \in U$ such that $\pi_v(u) \in N(v) \setminus U$

$$\sum_{u \in U} \sum_{w \in N(v) \setminus U} p^v_{u,w} \geq 1 \qquad \forall v \in V, \emptyset \neq U \subsetneq N(v). \tag{5.7}$$

The constraints 5.5, 5.6 and 5.7, together with 5.2, 5.3 and 5.4 form the first formulation of the problem.

5.3. Polynomially formulated models

Although the successor model is intuitive and links well to the rotation system π, due to the cyclicity of the permutation the number of constraints grows exponentially in terms of vertex degrees, as there is a constraint for every non-empty proper subset of neighbours of a vertex (see constraint 5.7). Because of that, other, polynomially sized formulations have been developed.

5.3.1. Index model

In the index model, a rotation system permutation is represented by a list of indices, through variables $q^v_{j,u}$ for $v \in V$, $u \in N(v)$ and $j \in [deg(v)]$. It is equal to 1 only if vertex u has index j in permutation around v. This means that if $\pi_v(u) = w$ and $q^v_{j,u} = 1$ then $q^v_{j \oplus_v 1,w} = 1$, where $j \oplus_v 1 = k \iff j + 1 \equiv k \ (mod\ deg(v))$. This model has been defined only for SAT [3], so we have to translate it to ILP ourselves.

First, we connect the variables to previously established variables of base. The SAT model states that for a given $v \in V$, $u, w \in N(v), u \neq w$ and $i \in [f]$, if there exists $j \in [deg(v)]$ such that u has index j in rotation around v and w has index $j \oplus_v 1$, then $c^i_{(u,v)} = c^i_{(v,w)}$. Written in logical form

$$\bigvee_{j \in [deg(v)]} (q^v_{j,u} = q^v_{j \oplus_v 1,w} = 1) \implies (c^i_{(u,v)} = c^i_{(v,w)}).$$

We can transform this into a conjunction of logical sentences through implication definition and De Morgan's laws

$$\neg \left(\bigvee_{j \in [deg(v)]} (q^v_{j,u} = q^v_{j \oplus_v 1,w} = 1) \right) \vee (c^i_{(u,v)} = c^i_{(v,w)})$$
$$\bigwedge_{j \in [deg(v)]} \left(\neg (q^v_{j,u} = q^v_{j \oplus_v 1,w} = 1) \right) \vee (c^i_{(u,v)} = c^i_{(v,w)})$$

$$\bigwedge_{j\in[deg(v)]} \left(\neg(q^v_{j,u} = q^v_{j\oplus_v 1,w} = 1) \vee (c^i_{(u,v)} = c^i_{(v,w)})\right)$$
$$\bigwedge_{j\in[deg(v)]} \left((q^v_{j,u} = q^v_{j\oplus_v 1,w} = 1) \implies (c^i_{(u,v)} = c^i_{(v,w)})\right).$$

This means, that we can divide this relation into multiple conditions

$$q^v_{j,u} = q^v_{j\oplus_v 1,w} = 1 \implies c^i_{(u,v)} = c^i_{(v,w)} \qquad \forall j \in [deg(v)],$$

which gives us the following ILP constraints

$$\begin{aligned} c^i_{(u,v)} &\geq c^i_{(v,w)} + (q^v_{j,u} + q^v_{j\oplus_v 1,w}) - 2 \quad \forall i \in [\bar{f}], v \in V, j \in [deg(v)] \quad \forall u, w \in N(v), u \neq w, \\ c^i_{(v,w)} &\geq c^i_{(u,v)} + (q^v_{j,u} + q^v_{j\oplus_v 1,w}) - 2 \quad \forall i \in [\bar{f}], v \in V, j \in [deg(v)] \quad \forall u, w \in N(v), u \neq w. \end{aligned} \tag{5.8}$$

Second, just like in $p^i_{u,w}$ variables, the $q^v_{j,u}$ variables have to be unique over j and u

$$\begin{aligned} \sum_{j\in[deg(v)]} q^v_{j,u} &= 1 \qquad \forall v \in V, u \in N(v) \\ \sum_{u\in N(v)} q^v_{j,u} &= 1 \qquad \forall v \in V, j \in [deg(v)]. \end{aligned} \tag{5.9}$$

These new constraints 5.8 and 5.9, together with the base constraints 5.2, 5.3 and 5.4 form the index formulation of the problem. As we assign indices to permutation vertices, the cyclicity of the permutation is assured, thus reducing the exponential number of constraints to a polynomial one.

5.3.2. Betweenness model

For the betweenness model, for every vertex $v \in V$, we introduce new variables $r^v_{a,b,c}$ for each $\{a, b, c\} \subseteq N(v)$. This variable is equal to 1 only if vertex b is between vertices a and c in the rotation around v. Notably

$$r^v_{a,b,c} \equiv r^v_{c,a,b} \equiv r^v_{b,c,a} \equiv \neg r^v_{a,c,b} \equiv \neg r^v_{b,a,c} \equiv \neg r^v_{c,b,a} \tag{5.10}$$

are equivalent, which, through substitution, lets us reduce the number of variables sixfold.

For the connection of this model to the base model, we use the constraint 5.5, where $p^v_{u,w}$ is equivalent to there being no vertices between u and w in permutation around v, or in logical terms

$$\bigwedge_{y\in N(v)\setminus\{u,w\}} \neg r^v_{u,y,w} \implies (c^i_{(u,v)} = c^i_{(v,w)}),$$

which gives us the following constraints

$$\begin{aligned} c^i_{(u,v)} &\geq c^i_{(v,w)} - \sum_{y\in N(v)\setminus\{u,w\}} r^v_{u,y,w} \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w \\ c^i_{(v,w)} &\geq c^i_{(u,v)} - \sum_{y\in N(v)\setminus\{u,w\}} r^v_{u,y,w} \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w. \end{aligned} \tag{5.11}$$

Then, it is only necessary to add a rule regarding the correctness of $r^v_{u,y,w}$ variables, which states that

$$(r^v_{u,w,x} = r^v_{u,x,y} = 1) \implies (r^v_{u,w,y} = r^v_{w,x,y} = 1).$$

This can be reformulated through implication definition and De Morgan's laws into two implications

$$\left((r^v_{u,w,x} = r^v_{u,x,y} = 1) \implies (r^v_{u,w,y} = 1)\right) \wedge \left((r^v_{u,w,x} = r^v_{u,x,y} = 1) \implies (r^v_{w,x,y} = 1)\right),$$

which gives us two sets of constraints

$$\begin{aligned} r^v_{u,w,x} + r^v_{u,x,y} - 1 \leq r^v_{u,w,y} \qquad & \forall v \in V, \{u,w,x,y\} \subseteq N(v), \\ r^v_{u,w,x} + r^v_{u,x,y} - 1 \leq r^v_{w,x,y} \qquad & \forall v \in V, \{u,w,x,y\} \subseteq N(v). \end{aligned} \tag{5.12}$$

This is properly defined for vertices with at least 4 neighbours. For vertices with a smaller degree, for which the minimum is 3, all the variables $r_{a,b,c}$ are connected through the equivalence relation 5.10 with each other, and every valuation produces a proper rotation system, without the need for any constraints.

6. Quantum annealing

Recently, a new method for solving optimization problems has been developed — quantum annealing. It is a method of utilizing quantum processes to find the minimum energy state of a given system. The method lets a user encode a problem into the energy of a system, and through quantum processes find its optimal value. This method is still being perfected, but it has the potential to revolutionise the optimization problem landscape. Due to the quantum processes involved with quantum annealing, such a method may be better suited to escaping local minima, which is a problem in traversing an optimization problem space, and the parallel nature of the model additionally speeds up the process of finding the optimum.

The main component of a quantum annealer is its processor, called a QPU (Quantum Processing Unit). It needs very low temperatures (under 20 mK) for it to function, as it relies on the phenomenon of superconductivity, in which current can pass through a medium without resistance. It consists of a number of qubits, bits that during the processing time can take any value between 0 and 1, and couplers, which connect certain qubits together. Those qubits and couplers form a simple connected graph, which is called the topology of the QPU.

6.1. The QUBO model

QUBO stands for *Quadratic Unconstrained Binary Optimization*. It is derivative of linear programming, although with some changes. The first term, *Quadratic*, means that variables can appear not only in linear form, but also as two variables multiplied together. The second term, *Unconstrained*, means that in the model, there are no constraints. There are other means by which we can ensure that the constraints would be satisfied. The remaining terms refer to the linear programming, in which all the variables can take values of either 0 or 1. The standard form for a QUBO model consists of an objective function

$$f = \sum_{i=1}^{n} a_i \cdot x_i + \sum_{i=1}^{n-1} \sum_{j=i}^{n} a_{ij} \cdot x_i x_j,$$

where $x_1, \ldots, x_n$ are binary variables and a_i and a_{ij} are constant parameters.

6.1.1. Reducing ILP to QUBO

To generate a QUBO model based on an ILP, we somehow need to remove all the constraints. However, they still need to be satisfied to make the model accomplish its goal. This can be achieved through utilizing the maximization of the objective function. If the model intents for the objective function to be maximized, then a term can be added that, if a constraint is satisfied then is equal to 0, and when it isn't the value is negative. This is called a penalty for the objective function. After such an operation, if the penalties are big enough, the model won't achieve maximum, unless all the constraints are satisfied.

6.1.2. Penalizing the model

Penalizing the model means converting its constraints into penalties for the objective function.

Equation constraints

Let $C_=$ be a constraint in the form of

$$C_= : \quad a_1 \cdot x_1 + \cdots + a_n \cdot x_n = a_{n+1},$$

where $x = [x_1, \ldots, x_n]$ is a vector of variables and $a = [a_1, \ldots, a_{n+1}]$ is a vector of parameters. In a more compact form it can be written as

$$C_= : \quad \sum_{i=1}^{n} a_i \cdot x_i = a_{n+1}.$$

The basic idea for converting such a constraint into a penalty $P_{C_=}$ is to move its free term to the left of the = sign and square the terms of the left side. This creates a following component

$$P_{C_=} = (\sum_{i=1}^{n} a_i \cdot x_i - a_{n+1})^2 =$$

$$= \sum_{i=1}^{n} a_i^2 \cdot x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j - 2a_{n+1} \sum_{i=1}^{n} a_i \cdot x_i + {a_{n+1}}^2.$$

One thing to note is that for a binary variable, squaring it doesn't change its value as $1^2 = 1$ and $0^2 = 0$, so we can transform the penalty even further

$$P_{C_=} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j + \sum_{i=1}^{n} (a_i^2 - 2a_{n+1} \cdot a_i) \cdot x_i + {a_{n+1}}^2.$$

After being squared, the value of $P_{C_=}$ will be zero if the constraint is satisfied, and will be greater than zero if the constraint is violated. Subtracting $P_{C_=}$ multiplied by a high enough constant λ from the objective function yields a proper form of a QUBO model, as all the variables appear alone or in pairs and the maximum of the function corresponds to a proper problem solution.

Inequality constraints

With constraints containing inequalities the situation is harder, as for a constraint

$$C_\leq : \quad a_1 \cdot x_1 + \cdots + a_n \cdot x_n \leq a_{n+1},$$

the model has to be penalized only when the left side exceeds a_{n+1}. The added component has to be equal to zero not only when the left side is equal to a_{n+1}, but also when it is lower than a_{n+1}. This can be done by adding a so called slack variable s, which can take non-negative integer values, to the left side. Then, the condition can be written as

$$C_\leq : \quad a_1 \cdot x_1 + \cdots + a_n \cdot x_n + s = a_{n+1}.$$

Next we can move the free term to the left side and square the left side to obtain a valid penalty

$$P_{C_\leq} = \left(\sum_{i=1}^{n} a_i \cdot x_i + s - a_{n+1}\right)^2 =$$

$$= \sum_{i=1}^{n} a_i^2 \cdot x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j + 2(s - a_{n+1}) \sum_{i=1}^{n} a_i \cdot x_i + s^2 - 2sa_{n+1} + {a_{n+1}}^2.$$

This, however, is not a QUBO penalty, as variable s is not binary. If we determine the maximum value which s can take and call it s_{max}, then s can be substituted with a set of binary variables $s_0, \ldots, s_k$ which form a binary representation for $s = \sum_{j=0}^{k} 2^j s_j$, where $k = \lfloor log_2(s_{max}) \rfloor$. Substituting, we obtain

$$P_{C_\leq} = \left(\sum_{i=1}^{n} a_i \cdot x_i + \sum_{j=0}^{k} 2^j s_j - a_{n+1}\right)^2 =$$

$$= \sum_{i=1}^{n} a_i^2 \cdot x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j + 2\left(\sum_{j=0}^{k} 2^j s_j - a_{n+1}\right) \sum_{i=1}^{n} a_i \cdot x_i +$$
$$+ (\sum_{j=0}^{k} 2^j s_j)^2 - 2a_{n+1} \sum_{j=0}^{k} 2^j s_j + {a_{n+1}}^2 =$$

$$= \sum_{i=1}^{n} a_i^2 \cdot x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j + \sum_{i=1}^{n} \sum_{j=0}^{k} 2^{j+1} a_i \cdot x_i s_j - 2a_{n+1} \sum_{i=1}^{n} a_i \cdot x_i +$$
$$+ \sum_{j=0}^{k} {2^{2j}} {s_j}^2 + \sum_{j=0}^{k-1} \sum_{i=j}^{k} 2^{j+i+1} \cdot s_j s_i - 2a_{n+1} \sum_{j=0}^{k} 2^j s_j + {a_{n+1}}^2.$$

As previously, squared variables can be changed into linear ones, thus resulting in the final penalty form

$$P_{C_\leq} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2a_i a_j \cdot x_i x_j + \sum_{i=1}^{n} \sum_{j=0}^{k} 2^{j+1} a_i \cdot x_i s_j + \sum_{j=0}^{k-1} \sum_{i=j}^{k} 2^{j+i+1} \cdot s_j s_i +$$
$$+ \sum_{i=1}^{n} (a_i^2 - 2a_{n+1} \cdot a_i) \cdot x_i + \sum_{j=0}^{k} (2^{2j} - 2^{j+1} a_{n+1}) \cdot s_j + {a_{n+1}}^2.$$

This, after multiplying by a λ, can be subtracted from the objective function.

The objective function containing all the constraints converted into penalties represents a proper QUBO model.

6.1.3. Representing QUBO as a graph

Each QUBO problem can be represented as a simple graph with some attributes associated with its nodes and edges. For a standard form of a QUBO model

$$f = \sum_{i=1}^{n} a_i \cdot x_i + \sum_{i=1}^{n} a_{ij} \cdot x_i x_j,$$

we create a graph of the model $P = (V, E)$, where V consists of n vertices $v_1, \ldots, v_n$, one for each variable. With each vertex v_i we associate the constant a_i, connected with the variable x_i. When it comes to the edges, there exists an edge between two vertices v_i and v_j only if the constant associated with multiplication of those variables, a_{ij}, is not equal to zero. Then, that constant parameter is connected to each corresponding edge. Whenever two variables x_i and x_j are multiplied, we will call that a connection of those variables, exactly due to their graph representation.

Such a graph representation of a QUBO model is very useful for determining, if an instance is solvable on a quantum annealer.

6.2. Determining QUBO solvability on a quantum annealer

For every quantum annealer, there is a simple graph G representing its architecture, called its topology. For a QUBO problem in a graph form P, if the graph P is a subgraph of G, it is solvable with the given quantum annealer. Additionally, some nodes of the annealer can be connected in such a way, that the resulting derived graph topology G' is the original graph topology G with some edges contracted. This approach is commonly used, as topologies tend to have vertices of equal, bounded degree, and this operation creates vertices with higher degrees. Such an operation lets the annealer solve problems with higher $\Delta(P)$, as a graph P cannot be a subgraph of G if $\Delta(P) > \Delta(G)$. In this manner, the question regarding if a problem can be solved on a given quantum annealer is reduced to asking if graph P is a minor of graph G. We will call this problem "embeddability of P in G".

In the following sections, different models will be analysed in terms of their embeddability. The bounds for their number of nodes and their degrees will be calculated.

Such a problem, where G is constant topology of a quantum annealer and P is a variable problem graph, and we want to know if P is embeddable in G, is NP-Hard [22]. As such, we will formulate the previous ILP models as QUBO problems and conduct a theoretical analysis of them in terms of complexity of the problem graphs they generate.

7. QUBO formulations of the minimum genus problem

To generate a QUBO from a binary ILP, we will follow the steps from the general method of penalizing the model (see chapter 6.1.2). As such, although constraints with equalities can be easily translated, for those containing inequalities it is beneficial to estimate a higher bound for the slack variable's value. Such an operation lowers the number of variables and connections between those in a graph generated from the QUBO, which can make embedding the problem in a given architecture easier or even possible.

For each of the constraints for the following models, we will transform them into ones with equality signs. Then we will perform an analysis of the number of variables in terms of the original graph's properties. After squaring the equation and multiplying it by a dedicated λ, we can determine how many variable pairs contain that variable, or in terms of the graph generated by the QUBO model, the degree of the variable, which for a variable x we will denote by $DEG(x)$, as to not mistake it with the degree of a vertex of the original problem graph. The squaring itself will not be done, as it is purely mechanical and there is no need for us to find the constants by which the variables are multiplied by, as only an analysis of the generated graph structure will be conducted.

In the following models, λ symbol will appear. For every penalty, this λ can be different, depending on the values that the penalty can generate and the real value of the graph genus. For simplicity, although the constants of the penalties may be different, they will be denoted by the same λ.

7.1. Base

The model base contains $\bar{f}$ variables x_i and $\bar{f} \cdot |A|$ variables c_a^i.

The first constraint 5.2 is an inequality, so slack variables are needed. To create them, we need to estimate how big can the slack be. The sum of variables c_a^i over $a \in A$ can be as big, as the biggest face of a graph. Theoretically, the biggest face could use all the graph's arcs, so we will need $\lfloor \log_2 |A| \rfloor + 1$ variables to be able to obtain at least $|A|$ as their sum. Thus, after adding $\lfloor \log_2 |A| \rfloor + 1$ variables s^i to each constraint, we get the first set of penalties in the form of

$$\lambda \left(\sum_{a \in A} c_a^i - 3x_i - \sum_{k=0}^{\lfloor \log_2 |A| \rfloor} 2^k s_k^i \right)^2 \qquad \forall i \in [\bar{f}]. \tag{7.1}$$

The second and third constraints, 5.3 and 5.4, are equations, so the transformation into

penalties can be easily preformed

$$\lambda \left(\sum_{i \in [\bar{f}]} c_a^i - 1 \right)^2 \qquad \forall a \in A, \tag{7.2}$$

$$\lambda \left(\sum_{a \in \delta^+(v)} c_a^i - \sum_{a \in \delta^-(v)} c_a^i \right)^2 \qquad \forall i \in [\bar{f}], v \in V. \tag{7.3}$$

At this point, we can determine the QUBO graph's structure in terms of variables x_i, as they don't appear in any more constraints in any model.

During the squaring of the term 7.1, a single variable x_i will be multiplied by c_a^i for each $a \in A$ and by every s_k^i for each $k \in \{0, \ldots, \lfloor \log_2 |A| \rfloor\}$. This means the QUBO graph will have $\bar{f}$ vertices x_i with

$$DEG(x_i) = |A| + \lfloor \log_2 |A| \rfloor + 1 = O(|E|).$$

We can do the same for the slack variables s_k^i, as each will be multiplied by one variable x_i, by c_a^i for each $a \in A$ and by every s_k^i for each $k \in \{0, \ldots, \lfloor \log_2 |A| \rfloor\}$ except for itself, which gives us $\bar{f}(\lfloor \log_2 |A| \rfloor + 1)$ variables s_k^i with

$$DEG(s_k^i) = |A| + \lfloor \log_2 |A| \rfloor + 1 = O(|E|).$$

7.2. Successor model

This model adds variables $p_{u,w}^v$, one for each $v \in V$ and $u, w \in N(v)$ such that $u \neq w$.

The first constraint 5.5 is an inequality, although, as it models a simple implication, for each inequality only one slack variable has to be added (resulting in $s_1^{i,v,u,w}$ and $s_2^{i,v,u,w}$). We obtain the following penalties

$$\begin{aligned} &\lambda \left(c_{(u,v)}^i - c_{(v,w)}^i - p_{u,w}^v + 1 - s_1^{i,v,u,w} \right)^2 \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w, \\ &\lambda \left(c_{(v,w)}^i - c_{(u,v)}^i - p_{u,w}^v + 1 - s_2^{i,v,u,w} \right)^2 \qquad \forall i \in [\bar{f}], v \in V \quad \forall u, w \in N(v), u \neq w. \end{aligned} \tag{7.4}$$

The next constraints, 5.6, are easier to translate as they are equation constraints, so we obtain

$$\begin{aligned} &\lambda \left(\sum_{w \in N(v) \setminus u} p_{u,w}^v - 1 \right)^2 \qquad \forall v \in V, u \in N(v), \\ &\lambda \left(\sum_{w \in N(v) \setminus u} p_{w,u}^v - 1 \right)^2 \qquad \forall v \in V, u \in N(v). \end{aligned} \tag{7.5}$$

In the last one, 5.7, the value of the left side can be as high as the size of the smaller subset. This gives us a bound for the slack sum of $\min(|U|, deg(v) - |U|) - 1$, which means

that every constraint needs $\lfloor \log_2(\min(|U|, deg(v) - |U|) - 1) \rfloor + 1$ slack variables. Thus, we obtain

$$\lambda \left(\sum_{u \in U} \sum_{w \in N(v) \setminus U} p^v_{u,w} - 1 - \sum_{k=0}^{\lfloor \log_2(\min(|U|, deg(v)-|U|)-1) \rfloor} 2^k s^{v,U}_k \right)^2 \quad \forall v \in V, \emptyset \neq U \subsetneq N(v), \tag{7.6}$$

with $s^{v,U}_k$ being the new slack variables.

Size analysis

We will analyse the resulting graph size by the variable families of it, with the first one being c^i_a. This variable is connected with every other c^i_a for the same face i from 7.1, which gives us $|A| - 1$ connections. From the same penalty, it is connected with variable x_i and variables s^i_k for $k \in \{0, \ldots, \lfloor \log_2 |A| \rfloor\}$, which, just as in the case of x_i from the same penalty, gives us $|A| + \log_2 |A| + 1$ connections in total.

The penalty 7.2 adds a connection of c^i_a with every c variable that has the same a, which generates $\bar{f} - 1$ new connections.

The penalty 7.3 doesn't add any new connections, as for a given i, all the variables c^i_a are already multiplied by each other.

For the penalty 7.4 a variable $c^i_{(u,w)}$ is already connected to the other variables from the same family. As for the other variables, $c^i_{(u,w)}$ appears in $deg(u) - 1 + deg(w) - 1$, as we can be discussing the neighbourhood either vertex u or w. For each of those there is a new connection with a p variable, an s_1 variable, and an s_2 variable, which gives us $3(deg(u) + deg(w) - 2)$ new connections.

All the penalties above add up to the degree of $c^i_{(u,w)}$, which is

$$DEG(c^i_{(u,w)}) = |A| + \log_2 |A| + \bar{f} + 3\, deg(u) + 3\, deg(w) - 6 = O(|E|),$$

as $\bar{f} \leq \left\lfloor \frac{2}{3}|E| \right\rfloor < |E|$.

Next we will discuss the variables $p^v_{u,w}$. From their definition, there are $\sum_{v \in V} deg(v)(deg(v) - 1) = O\left(\sum_{v \in V} deg(v)^2 \right)$ of them. Their first appearance is in the penalty 7.4, where there are 3 other variables for each $i \in [\bar{f}]$, which gives us $3\bar{f}$ connections.

The next penalty, 7.5, makes it, so $p^v_{u,w}$ is multiplied by every $p^v_{u_1,w_1}$ variable where $u_1 \in N(v)$ is different from u and by every p where $w_1 \in N(v)$ is different from w. This gives us $2(deg(v) - 2)$ new connections.

For the last penalty of the model, 7.6, it is harder to calculate the new connections of $p^v_{u,w}$. For a pair $p^v_{u_1,w_1}$ and $p^v_{u_2,w_2}$ there is a penalty where $u_1, u_2 \in U$ and $w_1, w_2 \in N(v) \setminus U$, where u_1,u_2, w_1 and w_2 are different pairwise. This, along with the previously analysed penalties tells us, that the only p variables from the neighbourhood of v with which $p^v_{u,w}$ is not connected, are $p^v_{y,u}$ and $p^v_{w,y}$ with $y \in N(v) \setminus \{u, w\}$, as well as $p^v_{w,u}$, which gives us $deg(v) \cdot (deg(v) - 1) - 2(deg(v) - 2) - 1 = deg(v)^2 - 3deg(v) + 3$ connections with other p variables.

Then, we need to examine in how many penalties from 7.6 a variable $p^v_{u,w}$ appears, as to see with how many slack variables it connects. The current bound for the slack variables,

although exact, is hard to grasp in simple terms, as it is irreducible in any meaningful way. However, if we fix the value obtainable by slack variables $s_k^{v,U}$ to the worst case scenario, which is $\lfloor deg(v)/2 \rfloor - 1$ (so the number of slack variables is fixed to $\lfloor \log_2 deg(v) \rfloor$) for every subset U, then the number of connections can be simplified further. As the number of slack variables is now constant for a given v, we can calculate how many times a single $p_{u,w}^v$ variable appears in a penalty and multiply it by the number of slack variables $\lfloor \log_2 deg(v) \rfloor$ to get the number of connections. This question can be reduced to "how many partitions of $N(v)$ into two non-empty sets exist, so that u is in one of them and w is in the other", which is the same as twice the number of subsets of $N(v) \setminus \{u, w\}$, as after the separation u can be added to either one, while w is added to the other. This gives us the number of new connections

$$2^{deg(v)-1} \cdot \lfloor \log_2 deg(v) \rfloor.$$

All together, the penalties mentioned above generate the following number of connections for each variable $p_{u,w}^v$

$$DEG(p_{u,w}^v) = 3\bar{f} + deg(v)^2 - 3deg(v) + 3 + 2^{deg(v)-2} \cdot \lfloor \log_2 deg(v) \rfloor = O\left(\bar{f} \cdot 2^{deg(v)} \log deg(v)\right).$$

The remaining variables to be examined are the slack variables in 7.4 and 7.6, which are simple to grasp, as a single slack variable appears in exactly one penalty. The variable $s_1^{i,v,u,w}$ appears $deg(v) \cdot (deg(v) - 1)$ times for each $i \in [\bar{f}]$ and $v \in V$, and is connected to 3 other variables. Thus, we have the number of variables $s_1^{i,v,u,w}$ equal to $\bar{f} \sum_{v \in V} deg(v) \cdot (deg(v) - 1) = O\left(\bar{f} \sum_{v \in V} deg(v)^2\right)$. With the variable $s_2^{i,v,u,w}$, the situation is analogous. The degree of both of them is

$$DEG(s_1^{i,v,u,w}) = DEG(s_2^{i,v,u,w}) = 3.$$

For the variable $s_k^{v,U}$ it is connected with others of its family with the same v and U (of which there are $\lfloor \log_2(\min(|U|, deg(v) - |U|) - 1) \rfloor$), and the variables $p_{u,w}^v$ in its penalty, for which the number is $|U| \cdot (deg(v) - |U|)$. All in all, there is a variable $s_k^{v,U}$ for each $v \in V$, $\emptyset \neq U \subsetneq N(v)$ and $k \in \{0, \ldots, \lfloor \log_2(\min(|U|, deg(v) - |U|) - 1) \rfloor\}$, and its degree is

$$DEG(s_k^{v,U}) = |U| \cdot (deg(v) - |U|) + \lfloor \log_2(\min(|U|, deg(v) - |U|) - 1) \rfloor.$$

This isn't very informative, so we can simplify the number of variables and their degrees by fixing the size of U to the worst case scenario, $|U| = deg(v)/2$. Thus, we obtain $\sum_{v \in V} (2^{deg(v)} - 2) \lfloor \log_2 deg(v) \rfloor$ variables with degrees

$$DEG(s_k^{v,U}) = \frac{deg(v)^2}{4} + \lfloor \log_2 deg(v) \rfloor = O(deg(v)^2).$$

All the variables, along with their degrees, can be seen in the Table 7.1.

Variable name	How many are there	Degree
x_i	$\bar{f}$	$O(\|E\|)$
c_a^i	$\bar{f} \cdot \|A\|$	$O(\|E\|)$
$p_{u,w}^v$	$O\left(\sum_{v \in V} deg(v)^2\right)$	$O\left(\bar{f} \cdot 2^{deg(v)} \log deg(v)\right)$
s_k^i	$O\left(\bar{f} \log \|A\|\right)$	$O(\|E\|)$
$s^{i,v,u,w}$	$O\left(\bar{f} \sum_{v \in V} deg(v)^2\right)$	3
$s_k^{v,U}$	$\sum_{v \in V} (2^{deg(v)} - 2) \lfloor \log_2 deg(v) \rfloor$	$O\left(deg(v)^2\right)$

Table 7.1: Variables of the successor model, with their number and degrees

7.3. Index model

This model adds variables $q_{j,u}^v$ for $v \in V$, $u \in N(v)$ and $j \in [deg(v)]$.

The first constraint, 5.8, is an inequality, so we need to create slack variables. The highest value the slack needs to sum up to is 3, so there is a need only for two slack variables per inequality, so for each $i \in [\bar{f}]$, $v \in V$, $j \in [deg(v)]$ and $u, w \in N(v), u \neq w$ we will create 4 variables: $s_{1,1}^{i,v,j,u,w}$, $s_{1,2}^{i,v,j,u,w}$, $s_{2,1}^{i,v,j,u,w}$ and $s_{2,2}^{i,v,j,u,w}$. The penalties take the following form

$$\lambda \left(c_{(u,v)}^i - c_{(v,w)}^i - (q_{j,u}^v + q_{j \oplus_v 1,w}^v) + 2 - s_{1,1}^{i,v,j,u,w} - 2s_{1,2}^{i,v,j,u,w}\right)^2 \quad \forall i \in [\bar{f}], v \in V, j \in [deg(v)] \quad \forall u, w \in N(v), u \neq w,$$

$$\lambda \left(c_{(v,w)}^i - c_{(u,v)}^i - (q_{j,u}^v + q_{j \oplus_v 1,w}^v) + 2 - s_{2,1}^{i,v,j,u,w} - 2s_{2,2}^{i,v,j,u,w}\right)^2 \quad \forall i \in [\bar{f}], v \in V, j \in [deg(v)] \quad \forall u, w \in N(v), u \neq w. \quad (7.7)$$

The other constraints are in the form of equations, so translating them to QUBO can easily be performed without adding any slack

$$\lambda \left(\sum_{j \in [deg(v)]} q_{j,u}^v - 1\right)^2 \quad \forall v \in V, u \in N(v),$$

$$\lambda \left(\sum_{u \in N(v)} q_{j,u}^v - 1\right)^2 \quad \forall v \in V, j \in [deg(v)]. \quad (7.8)$$

Size analysis

Starting with the variables c_a^i, they are connected to $|A| + \log_2 |A| + \bar{f}$ variables through penalties 7.1, 7.2 and 7.3. For the constraint 7.7, the reasoning is analogous to 7.4, with the addition of two more slack variables for every j between 1 and the vertex degree, so there are $4((deg(u)-1)^2 + (deg(w)-1)^2)$ new connections between $c_{(u,w)}^i$ and the slack variables. Additionally, there are q variables in the penalty, where $c_{(u,w)}^i$ is connected to every variable $q_{j,m}^w$ for every $j \in [deg(w)]$ and $m \in N(w)$, and $q_{j,m}^u$ for every $j \in [deg(u)]$ and $m \in N(u)$, which gives us $deg(u)^2 + deg(w)^2$ new connections.

All the penalties above add up to the degree of $c_{(u,w)}^i$, which is

$$DEG(c_{(u,w)}^i) = |A| + \log_2 |A| + \bar{f} + 4((deg(u)-1)^2 + (deg(w)-1)^2) + deg(u)^2 + deg(w)^2 =$$

$$= O(|E| + \Delta(G)^2),$$

with ΔG being the maximum degree of a vertex in the problem graph G.

Next we will discuss variable $q_{j,u}^v$. In the penalty 7.7, this variable connects with every $c_{(v,w)}^i$ and $c_{(w,v)}^i$ for $w \in N(v)$ and $i \in [\bar{f}]$, which gives us $2deg(v)\bar{f}$ connections. As for connecting with other $q_{j,u}^v$ variables, every $q_{j,u}^v$ is connected with $q_{j\oplus_v 1,w}^v$ and $q_{j\ominus_v 1,w}^v$ for every $w \in N(v)$, which results in $2deg(v)$ new connections. Regarding the connections with the slack variables, a single $q_{j,u}^v$ is connected to $s_{1,1}^{i,v,j,u,w}$,$s_{1,2}^{i,v,j,u,w}$,$s_{2,1}^{i,v,j,u,w}$ and $s_{2,2}^{i,v,j,u,w}$, and also to $s_{1,1}^{i,v,j\ominus_v 1,w,u}$,$s_{1,2}^{i,v,j\ominus_v 1,w,u}$,$s_{2,1}^{i,v,j\ominus_v 1,w,u}$ and $s_{2,2}^{i,v,j\ominus_v 1,w,u}$ for each $w \in N(v) \setminus \{u\}$ and $i \in [\bar{f}]$, generating $4(deg(v)-1)\bar{f})$ new connections.

As for the penalty 7.8, a variable $q_{j,u}^v$ connects to other q variables with the same $v \in V$ and $u \in N(v)$ of which there are $deg(v)$, and with other q variables with the same $v \in V$ and $j \in [deg(v)]$, which also gives $deg(v)$ connections. All of those connections are different from the one generated in 7.7, so there are $2deg(v)$ new connections.

In summary, there are $\sum_{v \in V} deg(v)^2$ variables $q_{j,u}^v$ with degrees

$$DEG(q_{j,u}^v) = 6deg(v)\bar{f} + 4deg(v) - 4\bar{f} = O(deg(v)\bar{f}).$$

Next are the last variables of this model, the slack variables $s^{i,v,j,u,w}$, for which the case is simple, as a single slack variable appears in only one penalty, so it is connected to exactly 5 other variables, which means that there are $4\bar{f}\sum_{v \in V} deg(v) \cdot deg(v) \cdot (deg(v)-1) = O\left(\bar{f}\sum_{v \in V} deg(v)^3\right)$ slack variables with degrees

$$DEG(s^{i,v,j,u,w}) = 5.$$

This is the last variable of the model, so we can group all the variables of the model, shown in the table 7.2.

Variable name	How many are there	Degree
x_i	$\bar{f}$	$O(\lvert E\rvert)$
c_a^i	$\bar{f}\cdot\lvert A\rvert$	$O\left(E+\Delta(G)^2\right)$
$q_{j,u}^v$	$\sum\limits_{v\in V} deg(v)^2$	$O(deg(v)\bar{f})$
s_k^i	$O\left(\bar{f}\log\lvert A\rvert\right)$	$O(\lvert E\rvert)$
$s^{i,v,j,u,w}$	$O\left(\bar{f}\sum\limits_{v\in V} deg(v)^3\right)$	5

Table 7.2: Variables of the index model, with their number and degrees

7.4. Betweenness model

This model adds variables $r_{a,b,c}^v$ for every $\{a,b,c\}\subseteq N(v)$.

For the first inequality, we need to add slack variables $s_k^{i,v,u,w}$ that add up to at least $deg(v)-1$ to match the r variables. Thus, we add $\lfloor\log_2 deg(v)\rfloor+1$ slack variables, obtaining the following penalty

$$\begin{aligned}
&\lambda\left(c_{(u,v)}^i-c_{(v,w)}^i+\sum_{y\in N(v)\setminus\{u,w\}} r_{u,y,w}^v-\sum_{k=0}^{\lfloor\log_2 deg(v)\rfloor} s_{1,k}^{i,v,u,w}\right)^2 && \forall i\in[\bar{f}], v\in V\\
& && \forall u,w\in N(v), u\neq w,\\
&\lambda\left(c_{(v,w)}^i-c_{(u,v)}^i+\sum_{y\in N(v)\setminus\{u,w\}} r_{u,y,w}^v-\sum_{k=0}^{\lfloor\log_2 deg(v)\rfloor} s_{2,k}^{i,v,u,w}\right)^2 && \forall i\in[\bar{f}], v\in V\\
& && \forall u,w\in N(v), u\neq w.
\end{aligned} \tag{7.9}$$

The second constraint is also made up of inequalities. However, the slack variables need to add up only to 2, so we need 2 slack variables $s^{v,u,w,x,y}$ per inequality, thus obtaining

$$\begin{aligned}
&\lambda\left(r_{u,w,x}^v+r_{u,x,y}^v-1-r_{u,w,y}^v+s_{1,1}^{v,u,w,x,y}+s_{1,2}^{v,u,w,x,y}\right)^2 && \forall v\in V,\{u,w,x,y\}\subseteq N(v),\\
&\lambda\left(r_{u,w,x}^v+r_{u,x,y}^v-1-r_{w,x,y}^v+s_{2,1}^{v,u,w,x,y}+s_{2,2}^{v,u,w,x,y}\right)^2 && \forall v\in V,\{u,w,x,y\}\subseteq N(v).
\end{aligned} \tag{7.10}$$

Size analysis

The first variable to analyse, $c_{(u,w)}^i$ is connected to $|A|+\log_2|A|+\bar{f}$ variables through penalties 7.1, 7.2 and 7.3. As for the penalty 7.9, The only new connections it forms are with

r variables and slack variables. Regarding the r variables, each $c^i_{(u,w)}$ variable connects with every r variable containing u and two other neighbours of w in its subscript, or w and two other neighbours of u in its subscript. From the equivalence relation 5.10 we know that the subscript variables (regardless of their order) uniquely define one r variable, so the number of new connections we get is $\frac{1}{2}deg(u)(deg(u)-1)+\frac{1}{2}deg(w)(deg(w)-1)$. For the slack variables, a $c^i_{(u,w)}$ variable connects to all the $s^{i,v,u,x}_k$ variables with the same i, v and u, and to all the $s^{i,v,x,w}_k$ variables with the same i, v and w, which gives us $deg(u)(\lfloor \log_2 deg(u) \rfloor + 1) + deg(w)(\lfloor \log_2 deg(w) \rfloor + 1)$ new connections. In total a variable $c^i_{(u,w)}$ has the degree of

$$DEG(c^i_{(u,w)}) = |A| + \log_2 |A| + \bar{f} + \frac{1}{2}(deg(u)^2 - deg(u) + deg(w)^2 - deg(w)) + \\ + deg(u)(\lfloor \log_2 deg(u) \rfloor + 1) + deg(w)(\lfloor \log_2 deg(w) \rfloor + 1) = O(|E| + \Delta(G)^2).$$

Next we will discuss the variables $r^v_{a,b,c}$. From the penalty 7.9, a single variable $r^v_{a,b,c}$ connects with $c^i_{(a,v)}$, $c^i_{(b,v)}$ $c^i_{(c,v)}$, $c^i_{(v,a)}$, $c^i_{(v,b)}$ and $c^i_{(v,c)}$ for every $i \in [\bar{f}]$, by the power of the equivalence relation, which creates $6\bar{f}$ connections. As for connecting with other r variables, an $r^v_{a,b,c}$ is connected to every r variable which contains any two of the vertices $\{a,b,c\}$ in its subscript. This generates $3(deg(v)-2)$ new connections. The last variable in this penalty is the slack variable. A single variable $r^v_{a,b,c}$ appears in 6 equations through the equivalence relation, so it is connected to $6(\lfloor \log_2 deg(v) \rfloor + 1)$ slack variables.

In the penalty 7.10, a variable $r^v_{a,b,c}$ connects to such other r variables that contain two of $\{a,b,c\}$ in their subscript. However, these connections are already taken care of in the analysis of 7.9, so the only new connections are formed with the slack variables. For a variable $r^v_{a,b,c}$ we get 4 connections with variables $s^{v,a,b,c,z}$ for every $z \in N(v) \setminus \{a,b,c\}$ if $r^v_{a,b,c}$ takes the role of $r^v_{u,w,x}$. If it takes the role of $r^v_{u,x,y}$, then we get 4 connections with $s^{v,a,z,b,c}$ for every $z \in N(v) \setminus \{a,b,c\}$. Next, it can take the role of $r^v_{u,w,y}$ connecting with two $s^{v,a,b,z,c}$ variables for every $z \in N(v) \setminus \{a,b,c\}$. Lastly, a variable can take the role of $r^v_{w,x,y}$, creating two connections with variables $s^{v,z,a,b,c}$ for every $z \in N(v) \setminus \{a,b,c\}$. In total, this gives us $12(deg(v)-3)$ new connections with slack variables.

Summing it all up, there are $\sum_{v \in V} \frac{1}{6}deg(v)(deg(v)-1)(deg(v)-2) = O\left(\sum_{v \in V} deg(v)^3\right)$ $r^v_{a,b,c}$ variables with degrees

$$DEG(r^v_{a,b,c}) = 6\bar{f} + 3(deg(v)-2) + 6(\lfloor \log_2 deg(v) \rfloor + 1) + 12(deg(v)-3) = \\ = 6\bar{f} + 15deg(v) + 6\lfloor \log_2 deg(v) \rfloor - 36 = O(\bar{f} + deg(v)).$$

For the slack variables, first we will discuss the variable $s^{i,v,u,w}_{1,k}$. It appears only in one equation, so it connects only with the variables in one penalty, resulting in $2 + (deg(v) - 2) + \lfloor \log_2 deg(v) \rfloor$ connections. For the variable $s^{i,v,u,w}_{2,k}$, the situation is analogous. This means there are $\bar{f} \sum_{v \in V} deg(v)(deg(v)-1)(\lfloor \log_2 deg(v) \rfloor + 1) = O\left(\bar{f} \sum_{v \in V} deg(v)^2 \log deg(v)\right)$ $s^{i,v,u,w}_k$ variables with

$$DEG(s^{i,v,u,w}_{1,k}) = DEG(s^{i,v,u,w}_{2,k}) = deg(v) + \lfloor \log_2 deg(v) \rfloor = O(deg(v)).$$

The other slack variable, $s^{v,u,w,x,y}$, connects only to the 4 other variables in its penalty. There are $\sum_{v \in V} \frac{1}{24} deg(v)(deg(v)-1)(deg(v)-2)(deg(v)-3) = O\left(\sum_{v \in V} deg(v)^4\right)$ $s^{u,w,x,y}$ variables with degree

$$DEG(s^{u,w,x,y}) = 4.$$

All the variables of this model are shown in the table 7.3.

Variable name	How many are there	Degree
x_i	$\bar{f}$	$O(\|E\|)$
c_a^i	$\bar{f} \cdot \|A\|$	$O(E + \Delta(G)^2)$
$r_{a,b,c}^v$	$O\left(\sum_{v \in V} deg(v)^3\right)$	$O(\bar{f} + deg(v))$
s_k^i	$O\left(\bar{f} \log \|A\|\right)$	$O(\|E\|)$
$s_k^{i,v,u,w}$	$O\left(\bar{f} \sum_{v \in V} deg(v)^2 \log deg(v)\right)$	$O(deg(v))$
$s^{v,u,w,x,y}$	$O\left(\sum_{v \in V} deg(v)^4\right)$	4

Table 7.3: Variables of the betweenness model, with their number and degrees

8. Practical framework

As can be seen in the previous chapters, formulation size of the minimum genus of a graph problem can grow rapidly with the graph size. This is why preprocessing of the input graph is crucial to faster problem-solving.

8.1. Preprocessing

To understand the basic theorem that lets us shrink the problem size, some graph concepts need to be explained. A graph G is connected, if for every two vertices of G, there exists a subgraph of G that is a path between those vertices. A path between two vertices u,v is a simple graph, in which vertices can be ordered in such a way, that an edge exists between two vertices only if they are two consecutive vertices in that order [33]. A connected graph is biconnected if, after deleting any vertex and its incident edges from it, it is still connected. Any graph can be divided into biconnected components, which are its maximum subgraphs that are biconnected.

Then, the theorem states, that the genus of a graph is additive over biconnected components [2, Theorem 1]. This means we can divide the problem graph into its biconnected components, also called blocks, and then preform calculations for each one of them separately.

The next property, measuring how connected the graph is, is triconnectness. A graph is triconnected if it is connected after removing any two vertices along with their incident edges from it. Although the genus of a graph isn't additive over triconnected components, dividing a graph into those is beneficial in other ways, explored in the following sections. A standard way to represent a biconnected graph divided into its triconnected components is called an SPQR tree.

8.1.1. SPQR trees

An SPQR tree is a tree. A tree is a simple, connected graph $G = (V, E)$ with exactly $|V| - 1$ edges. A result of that is, that a tree contains vertices $v \in L \subseteq V$ such that $deg(v) = 1$, which are called leaves. Any leaf can be deleted from a tree, and the graph will still be a tree.

In an SPQR tree, each one of the vertices, also called a node, is associated with a connected graph. The graph's nodes can be one of four types, for which the capital letters SPQR stand,

- S node stands for a cycle, which is a path between vertices s and t comprised of at least 3 vertices, with an additional edge between s and t,
- P node stands for a multigraph with two vertices and a number of edges between them,
- Q node stands for a graph with two nodes and an edge between them. It is necessary for the case of an SPQR tree of a graph consisting of only one edge and will not appear in this thesis,

- R node stands for a triconnected component.

Any biconnected graph can be represented by an SPQR tree, and there is a one to one correspondence between a graph and its SPQR tree. Additionally, for any SPQR tree T and its leaf l, after removing l from T, the obtained SPQR tree T' can be translated into a valid biconnected graph[1].

Given a leaf of an SPQR tree, we can check if the graph associated with that leaf is planar. If a leaf is an S node or a P node, then it is automatically planar. For the R nodes, checking for planarity of any graph $G = (V, E)$ can be done in $O(|V|)$ time [5]. If a leaf happens to be planar, it can be deleted from the SPQR tree. After a small modification to its neighbouring vertex, this operation, after translating the tree back into a graph, results in the original graph with the planar component substituted by a single edge. Importantly, such an operation preserves the genus of the graph [3]. We can continue with the deletion of planar leaves of an SPQR tree, until all the leaves have non-planar graphs associated with them, or the tree is empty, in which case the genus of the original graph was 0. Such an operation is called a non-planar core reduction of a graph [7]. After this reduction, if the graph obtained from the pruned SPQR tree contains any edges $\{u, w\}$ such that $deg(u) = 2$ or $deg(w) = 2$, we can contract them, until no such edges exist. Then we are left with a biconnected graph with all the maximal triconnected planar components replaced by single edges, and containing only vertices with at least three neighbours.

8.1.2. Computational complexity

Splitting a graph $G = (V, E)$ into biconnected components can be done in linear time $O(|V| + |E|)$ [18]. Then, an SPQR tree can be generated also in linear time $O(|V| + |E|)$ [16]. Testing for planarity of a graph can be done in $O(n)$ time, with n being the number of its vertices [5]. A non-planar core reduction can also be done linearly, in $O(|V| + |E|)$ [7]. Edge contraction will take at most $O(|E|)$ time, which all together yields a linear $O(|V| + |E|)$ algorithm for splitting the problem graph into subproblems and then reducing them to smaller sizes.

8.2. The problem in practice

8.2.1. Current landscape

Throughout the years, two main approaches of solving the minimum genus of a graph problem have been developed,

- backtracking algorithms, which search through the problem space, and backtrack whenever they see that the current route won't yield better results,
- ILP and SAT models, which statically simulate the face tracing algorithm and optimize for the highest number of faces.

[1] As every node of an SPQR tree has a graph associated with it, the node removal requires some additional steps (see [7])

The most widely known implementation of an algorithm for finding the minimum genus of a graph can be found in SageMath mathematics software [28]. It is a backtracking algorithm, and although it is very simple to use due to its compatibility with Python [31] and its graph libraries such as NetworkX [17], it is very slow, which even the authors acknowledge, stating that calculating the genus of a K_7 graph "may take a few days" [29]. Another backtracking algorithm, written in Java, was recently used by the graph database House of Graphs [9], which ran faster than the SageMath version, although it might have been already replaced by an even faster one [6].

The next improvements came in the form of models for Integer Linear Programming and Boolean Satisfiability solvers [3, 8]. Those models were discussed previously, although there are some improvements to them which are out of the scope of this thesis. Those methods proved to be more viable than the contemporary ones.

More recently, better backtracking algorithms have been developed. The first one, called *multigenus*, is the currently best performing algorithm for graphs that have vertices with degrees higher than 5 [6]. The more recent second one, called *PAGE*, is better suited for graphs of the highest degree 5 or lower, with the time complexity of $O(2^{n^2+3n}/n^{(n+1)})$ [23]. Programs for both of those algorithms, written in C, outperform the previously established ILP and SAT models and are available publicly, released by their authors.

8.2.2. Reality of quantum computing

Creating a quantum computer is hard and costly. As such, not many such computers exist, and the sizes of their QPUs are limited. In this paper, we will focus on the quantum annealers produced by the company leading in the analogue quantum computations space, D-Wave Systems [10]. Currently, D-Wave has developed two topologies for their quantum annealers, Chimera topology and Pegasus topology, and is currently developing the third one, Zephyr topology. All the topologies are well documented and publicly available. In this thesis, we will focus on their characteristics (see Table 8.1), which will be useful in the analysis of viability of the ILP models.

Topology	Qubits (vertices)	Couplers (edges)	Connectivity (degree)
Chimera	2000+	6000+	6
Pegasus	5000+	40000+	15
Zephyr	1200+	11000+	20

Table 8.1: Characteristics of different D-Wave QPU topologies[2], with connectivity of the topology being the degree of any of its vertices

For any QPU there exists a set of qubits which are not working properly and are excluded from calculations, so the real number of qubits and couplers for a given QPU may be lower.

[2]source: https://www.dwavesys.com/solutions-and-products/systems/

8.3. Problem embeddability in QPU

Before attempting to embed a QUBO problem into a QPU architecture, it is beneficial to check if such an operation is even possible. A quick way to check it is to see, if for a problem graph $P = (V, E)$, $|V|$ is lower than the topology qubits and $|E|$ is lower than the number of couplers, as otherwise the problem cannot possibly be embedded.

Three representative graphs have been chosen for such a test, K_5 and $K_{3,3}$, as they are the smallest non-planar graphs, and the Petersen graph, which has K_5 as a minor but all of its vertices are of degree 3 (see Figure 8.1).

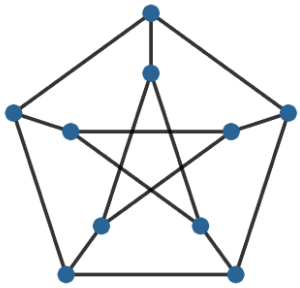

Figure 8.1: The Petersen graph, similar in structure to K_5 but with lower vertex degrees

The graphs are irreducible by the previously mentioned preprocessing steps, as they are all triconnected and non-planar. The exact properties of the problem graphs generated using different ILP models can be found in the table 8.2[3][4].

Graph	Successor model			Index model			Betweenness model		
	$\|V\|$	$\|E\|$	$\Delta(G)$	$\|V\|$	$\|E\|$	$\Delta(G)$	$\|V\|$	$\|E\|$	$\Delta(G)$
$K_{3,3}$	324	1467	32	1422	7587	86	294	1539	54
K_5	820	3950	42	5010	26530	172	770	4950	97
Pet	840	5085	46	3870	21885	118	790	5325	90

Table 8.2: Properties of model graphs for different representative graphs

Just looking at the number of vertices and edges, the index model cannot be embedded on Chimera and Zephyr topologies. Apart from that, all the models for all the graphs should be embeddable on each topology. However, there is another problem, connected to vertex degrees. All the maximum vertex degrees in all the models exceed the capabilities of the QPUs.

[3]For all the following experiments, an upper bound for the number of faces $\bar{f} = min(\lfloor \frac{2}{3}|E| \rfloor, |E| - |V|)$ was used.

[4]Specification on how to recreate the test results can be seen in Appendix A

Although some edges of the topology can be modified in such a way, that two qubits become conjoined, practically contracting an edge in a topology, this operation in turn removes some variables and edges.

We can simulate this contraction in reverse, by splitting every vertex v in the problem graph into $\lceil deg(v)/\Delta(T) \rceil$ vertices, with T being the QPU topology, giving us a lower bound for the number of vertices needed in a topology to embed the graph, which can be seen in the table 8.3.

Graph	Successor model			Index model			Betweenness model		
	$\|V_C\|$	$\|V_P\|$	$\|V_Z\|$	$\|V_C\|$	$\|V_P\|$	$\|V_Z\|$	$\|V_C\|$	$\|V_P\|$	$\|V_Z\|$
$K_{3,3}$	657	420	366	2817	1896	1785	549	354	345
K_5	1650	1060	1055	9740	6870	6325	1670	1070	1025
Pet	2065	1330	1180	7975	5380	4900	1975	1120	1110

Table 8.3: Lower bound for the number of vertices for each topology and model, with $|V_C|$, $|V_P|$, $|V_Z|$ being the bound for the number of vertices of Chimera, Pegasus, and Zephyr topologies respectively

Although simple, this bound is sometimes more than double the initial number of vertices. This, along with the table 8.2, also shows how inefficient the index model is compared to the other ones for graphs with low maximum degree. It can be deduced from the theoretical analysis that as the graph vertices rise in degree, the index model will prove more viable, as $\sum_{v \in V} 2^{deg(v)}$ will eventually be bigger than $\sum_{v \in V} deg(v)^3$.

The bound also proves useful, as there is no efficient way of testing if a given graph H is a minor of graph G, because, as it was previously said, this problem is NP-hard. Its hardness was also proven for the D-Wave QPU topologies specifically [22]. The company provides the program which is used for embedding problem graphs into topologies [11], although it is heuristical in nature and can not say with certainty if a graph is embeddable, especially if the graph's parameters are close to the topology ones.

8.4. Testing for embeddability

Using the bound mentioned previously, a pipeline was created to check, if a graph could be (at least theoretically) embedded in a QPU topology. All the code mentioned below was written in Python3 [31], with the usage of the NetworkX library [17] and the SageMath library [28]. For the tests of correctness, the genus calculating algorithm *multigenus* was used [6][5].

[5]The code is available under *multi_genus.c*: https://caagt.ugent.be/cubic-bip-nonham/

Given an input graph G, the goal was to obtain QUBO problem graphs P_S, P_I and P_B for each of the previously explored models, successor model, index model and betweenness model respectively.

8.4.1. Graph preprocessing

The first step was to preprocess the given graph into a smaller, yet equivalent version in terms of its genus. The graph was split into its biconnected components, using the *blocks_and_cut_vertices* function from sage library. Let B be such a component. All the following actions were done for every biconnected component.

For a biconnected graph B, a planarity test was conducted using the sage library function implementation of the *planarity by edge addition* algorithm by Boyer and Myrvold [5]. This was done before generating an SPQR tree of $B = (V, E)$, as in case of a planar graph, one algorithm with time complexity $O(|V|)$ could be used, instead of creating an SPQR tree and checking for planarity of each of the graphs associated with its nodes, which would take $O(|V| + |E|)$.

The next step was to generate an SPQR tree T of the graph B. This was done using the *spqr_tree* function from the sage library. Then, T was pruned, which means that while it had any leaves that were planar (S or P nodes, or in case of R nodes those, that passed the *planarity by edge addition* test) they were deleted, with a small modification to their neighbours[6]. By keeping the leaves in a queue and adding a neighbour of a deleted leaf to that queue if its degree became 1, this could be done in linear time in terms of vertices of T.

After the tree T was pruned, it was turned into the corresponding graph. Then the last step of the preprocessing consisted of contracting the edges incident to vertices with degree equal to two, until no such vertices existed.

As a result, a list L of biconnected non-planar cores $L = (C_1, \ldots, C_n)$ was obtained. The steps traced above can be seen in the form of a pseudocode in algorithm 8.1.

8.4.2. Generating models

After obtaining the non-planar core, QUBO models could be generated. For each model, a graph was created, where the vertices corresponded to the variables of the QUBO model. Then, for each penalty of a given model, the constants correlated with each variable were put in an array, after which all of the elements in the array were multiplied pairwise, simulating squaring of a penalty. If two variables were multiplied together, an edge was created between them with the value of the multiplication. If the edge already existed, then only the new value was added. Whenever a variable was multiplied by itself, the result would be assigned to its vertex, or, if it already had a value, added to it. If any constant was multiplied by itself, the result was stored in a variable separate from the graph, called the *offset*. This was conducted for every penalty of a given model, yielding a proper QUBO graph.

8.4.3. Determining embeddability

To answer whether a graph could be embedded in a topology T, the previously mentioned bound was calculated. For the set of vertices V, the number of the vertices after decontraction

Algorithm 8.1: Non-planar core reduction

```
Data: Graph G,
Result: list L of biconnected non planar graphs L = (C_1, ..., C_n)
L ←empty list
foreach biconnected component B of G do
    if B is not planar then
        T ← SPQR tree of B
        q ← queue of leaves of T
        while q is not empty do
            leaf ← q.pop()
            if leaf is an S or P node OR graph associated with leaf is planar then
                parent ← neighbour of leaf
                remove leaf from T
                modify parent
                if deg(parent) = 1 then
                    q.push(parent)
        C ← graph generated from T
        while edge e = (u, w) such that deg(u) = 2 or deg(w) = 2 exists do
            contract e
        add C to L
```

was set to

$$|V'| = \sum_{v \in V} \lceil deg(v)/\Delta(T) \rceil ,$$

and the number of edges of the graph was set to

$$|E'| = |E| + (|V'| - |V|),$$

as every decontracted vertex generates at least one edge.

8.4.4. Results

For the tests, a widely used graph dataset *Rome* was used, containing graphs appearing in real life scenarios and commonly used for graph genus testing [1, 12]. Out of 11534 graphs in this dataset, 3281 were confirmed to be planar, so they were omitted in the following analysis. For the other 8253 graphs, they were separated into their non-planar cores, and then for each a QUBO graph was generated. After that, a bound for the vertices and edges after decontraction was established. The results of how many graphs remained below the topologies' specifications can be seen in the table 8.4.

The results show that not many graphs could be embedded in the topologies, even though the bound is very generous, as it omits the specificity of the topology structure, and the

[6] More details about the modification can be found in [7]

Topology	Successor model	Index model	Betweenness model
Chimera	177	0	98
Pegasus	1223	94	1223
Zephyr	214	0	214

Table 8.4: Results indicating how many QUBO problem graphs of graphs from *Rome* out of 8235 could theoretically be embeddable in given topologies

graphs themselves aren't that big, as even before the non-planar core reduction they had no more than 100 vertices each, with the ratio $|E|/|V| \approx 1.3$.

It can be easily noticed that the successor model, although exponential in graph size, gives the best results. However, this may be due to the specificity of the graph dataset used, as other models should shine in scenarios where degree is high.

To see the models' performance on graphs with higher degrees, two graph families were chosen as representatives, K_n and $K_{3,n}$. Both of the graphs for any $n \geq 5$ are non-planar, as they contain K_5 and $K_{3,3}$ as subgraphs respectively. Another reason for this choice was that they represent two sides of a spectrum, with K_n having high maximum degree and high average degree, and $K_{3,n}$ having high maximum degree and low average degree. As simply tracking the number of variables isn't informative of the performance of the model, because it omits the vertex degree, the experiment was conducted with a decontraction into the Chimera topology[7]. For each model and each n, the number of vertices was tracked. The results can be seen in Table 8.5.

The test shows, that for $K_{3,n}$ the betweenness model is smaller than the successor model for any n higher than 8. For K_n the same holds for $n > 11$. This means that for graphs with high maximum degree, opting for the polynomial model creates a smaller formulation. For K_n, where the average degree is high, n had to be higher than for $K_{3,n}$ for the betweenness model to be the smaller one, which was expected due to the impact of the vertex degrees in its theoretical size evaluation. Additionally, for $K_{3,n}$ also the index model starts to be smaller than the successor model from $n = 14$ onwards. However, it doesn't get smaller than the betweenness model. This can also be deduced from the theoretical analysis, as for a graph $G = (V, E)$

$$\bar{f} \sum_{v \in V} deg(v)^3 > \sum_{v \in V} deg(v)^4,$$

with the used $\bar{f} = min(\frac{2}{3}|E|, |E| - |V|)$. The index model may prove more viable if a better bound for the number of faces is found.

[7]All the topologies gave similar comparative results, so we can choose any one of them without the loss of generality

n	K_3, n		
	$\lvert V^S\rvert$	$\lvert V^I\rvert$	$\lvert V^B\rvert$
8	<u>25004</u>	151856	29324
9	46071	246375	<u>45765</u>
10	93590	378770	<u>68120</u>
⋮		⋮	
13	1020149	1111019	<u>205226</u>
14	2338022	<u>1504160</u>	<u>271670</u>
15	5355951	<u>1995165</u>	<u>349515</u>

n	K_n		
	$\lvert V^S\rvert$	$\lvert V^I\rvert$	$\lvert V^B\rvert$
11	<u>447174</u>	2744964	475224
12	1007072	4913084	<u>939344</u>
13	2319304	8216312	<u>1530646</u>

Table 8.5: The number of vertices of $K_{3,n}$ and K_n for different N and different models, with $|V^S|$,$|V^I|$ and $|V^B|$ being the numbers of vertices in successor, index, and betweenness models respectively. Values were underlined whenever they were the lowest, or were lower than the ones in the successor model

9. Conclusions

The problem of finding the minimum genus of a graph is big in terms of its ILP formulation, regardless of the model, and the tests have shown that most of the formulations exceed the current capabilities of quantum annealers. This, however, may be improved in the future, as new topologies are being developed. Out of all the models, the exponential successor model, although big in its theoretical analysis, proved to be the most viable model for the test dataset. As for the other models, one should choose the betweenness model for graphs of high maximum degree, because even in dense graphs the polynomial formulation beats the exponential one. When it comes to the index model, it can be seen that there is a point at which it becomes smaller than the successor model in the table for $K_{3,n}$. For the denser graphs of K_n type, we might deduce that a similar point exists too, as the rate at which every entry is bigger than the previous one, is smaller in the index model than in the successor model[1]. If some other tests are developed in the future, that show any superiority of the index model over the betweenness model, then it might be considered as a viable option. However, for now, when it comes to the smallest model for graphs of higher maximum degree, the betweenness model is the best choice.

A different aspect of testing for viability of a formulation, is the performance testing. Different models may give different results in terms of computation time, or how many times a calculation has reached the optimal solution (as quantum computers are probabilistic in their nature). Those tests are necessary to determine if any model can be ruled out as worse than the others. However, quantum computers, due to their novelty, complicated manufacturing process and maintenance, are expensive, and as such, very costly to use. One would need substantial funds to perform tests which are thorough enough to yield proper results, which is why such test are out of the scope of this thesis. This is also why the size and embeddability of a model was chosen as a criterium for exploration, as it is important for the problem, but can be done without using a quantum annealer itself.

There exists another way of lowering the size of the problem, called relaxation. It involves removing certain conditions or constraints from an ILP model, which still sometimes can result in a proper solution or a good approximation, with less computation. This, however, was not explored in this thesis, as for the given problem it is hard to find constraints which could be dropped to still obtain a valid solution, as all the constraints in the ILP models ensured that a solution represents a real embedding. This means that the tests for checking which constraints or variables may be omitted have to be empirical, and are therefore a subject for further research.

As for other ways of exploring the problem using quantum operations, D-Wave Systems offers Hybrid Solvers, which provide a hybrid between QPUs, CPUs and GPUs, merging classical computers with the quantum ones. It is yet to be shown how the genus models preform on such solvers, and how it compares to QPU-only methods, especially as such solvers

[1]This point is hypothesised, as the calculations have timed out for higher n

enable the user to exceed the limits of the properties of QPU topology.

When it comes to the exploration of the ILP models themselves, other, more complicated ILP models, such as presented by Chimani and Wiedera [8], have been developed. Those may also be explored as QUBO instances, and may generate better results, just as they did for classical solvers. This thesis could be used as a basis for such transformations.

Lastly, the theory regarding the minimum genus of a graph problem may also be developed. Currently, the problem space isn't well explored and needs further research [3], as examining it might create new heuristics for the problem, or generate simpler models. Other aspect of developing the theoretical field of the problem is the preprocessing that can be done. At the moment, any graph can be broken into biconnected components, and non-planar core reduction can be performed on those. Further research in the field might result in a theorem for additivity of the problem for triconnected components, other reductions in the SPQR tree or even different approaches to preprocessing.

In conclusion, the field of finding the minimum genus of a graph is constantly evolving, and new algorithms are being developed. Although at the moment the ILP models aren't better than the backtracking ones, with more research in the field, quantum annealers may give the ILP models an upper hand again.

Bibliography

[1] G. D. Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5-6):303–325, Apr. 1997.

[2] J. Battle, F. Harary, and Y. Kodama. Additivity of the genus of a graph. *Bulletin of the American Mathematical Society*, 68(6):565–568, 1962.

[3] S. Beyer, M. Chimani, I. Hedtke, and M. Kotrbčík. A practical method for the minimum genus of a graph: Models and experiments. In *Proceedings of the 15th International Symposium on Experimental Algorithms - Volume 9685*, SEA 2016, page 75–88, Berlin, Heidelberg, 2016. Springer-Verlag.

[4] T. J. Boothby. *Rotation systems: Theory and application*. PhD thesis, Bachelor's thesis, University of Washington, 2007.

[5] M. W. J. Boyer, John M. On the cutting edge: simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[6] G. Brinkmann. A practical algorithm for the computation of the genus, 2020.

[7] M. Chimani and C. Gutwenger. Non-planar core reduction of graphs. *Discrete Mathematics*, 309(7):1838–1855, 2009. 13th International Symposium on Graph Drawing, 2005.

[8] M. Chimani and T. Wiedera. Stronger ILPs for the Graph Genus Problem. In M. A. Bender, O. Svensson, and G. Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[9] K. Coolsaet, S. D'hondt, and J. Goedgebeur. House of graphs 2.0. https://houseofgraphs.org, Last accessed 15 November 2024.

[10] D-Wave Quantum Systems Inc. dwave. https://www.dwavesys.com, Last accessed 6 December 2024.

[11] D-Wave Quantum Systems Inc. *minorminer Documentation*. Release 0.2.6, Last accessed 15 November 2024.

[12] S. Di Bartolomeo, E. Puerta, C. Wilson, T. Crnovrsanin, and C. Dunne. A collection of benchmark datasets for evaluating graph layout algorithms. Under submission to Graph Drawing Posters, 2023. https://visdunneright.github.io/gd_benchmark_sets/, Last accessed 10 December 2024.

[13] D. P. Dobkin and S. P. Reiss. The complexity of linear programming. *Theoretical Computer Science*, 11(1):1–18, 1980.

[14] I. S. Filotti, G. L. Miller, and J. H. Reif. On determining the genus of a graph in o(v o(g)) steps(preliminary report). *Proceedings of the eleventh annual ACM symposium on Theory of computing*, 1979.

[15] J. Gross and T. Tucker. *Topological Graph Theory*. Dover books on mathematics. Dover Publications, 2001.

[16] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In J. Marks, editor, *Graph Drawing*, pages 77–90, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[17] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[18] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[19] K. ichi Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.

[20] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[21] T. Korhonen, M. Pilipczuk, and G. Stamoulis. Minor containment and disjoint paths in almost-linear time, 2024.

[22] E. Lobe and A. Lutz. Minor embedding in broken chimera and derived graphs is NP-complete. *Theoretical Computer Science*, 989, 2024.

[23] A. Metzger and A. Ulrigg. An efficient genus algorithm based on graph rotations, 2024.

[24] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.

[25] B. P. Mull, R. G. Rieper, and A. T. White. Enumerating 2-cell imbeddings of connected graphs. *Proceedings of the American Mathematical Society*, 103(1):321–330, 1988.

[26] W. Myrvold and W. Kocay. Errors in graph embedding algorithms. *Journal of Computer and System Sciences*, 77(2):430–438, 2011. Adaptivity in Heterogeneous Environments.

[27] N. Robertson and P. Seymour. Graph minors. xx. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte.

[28] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5)*, 2022. https://www.sagemath.org, Last accessed 15 November 2024.

[29] The Sage Development Team. Sage documentation: Genus. https://doc.sagemath.org/html/en/reference/graphs/sage/graphs/genus.html#module-sage.graphs.genus, Last accessed 6 December 2024.

[30] C. Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[31] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[32] L. Wan. The genus of a graph: A survey, appendix a. graphs with known genera. *Symmetry*, 15(2), 2023.

[33] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

A. Tests execution

The tests mentioned in the thesis can be executed using the attached zip archive, with the file *README.md* containing the code execution instructions.