HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

Finding Periodic Apartments: A Computational Study of Hyperbolic Buildings

MSc thesis

Jarkko Savela

Supervisors
Associate Professor Matti Järvisalo
Docent Emilia Oikarinen

Examiner
Professor Juha Kontinen

June 18, 2020

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Mathematics and Statistics

Tekijä — Författare — Author

Jarkko Savela

Työn nimi — Arbetets titel — Title

Finding Periodic Apartments: A Computational Study of Hyperbolic Buildings

Oppiaine — Läroämne — Subject

Mathematics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	June 2020	67 pp.

Tiivistelmä — Referat — Abstract

This thesis presents a computational study of a fundamental open conjecture in geometric group theory using an intricate combination of Boolean Satisfiability and orderly generation. In particular, we focus on Gromov's subgroup conjecture (GSC), which states that "each one-ended hyperbolic group contains a subgroup isomorphic to the fundamental group of a closed surface of genus at least 2". Several classes of groups have been shown to satisfy GSC, but the status of non-right-angled groups with regard to GSC is presently unknown, and may provide counterexamples to the conjecture. With this in mind Kangaslampi and Vdovina constructed 23 such groups utilizing the theory of hyperbolic buildings [International Journal of Algebra and Computation, vol. 20, no. 4, pp. 591–603, 2010], and ran an exhaustive computational analysis of surface subgroups of genus 2 arising from so-called periodic apartments [Experimental Mathematics, vol. 26, no. 1, pp. 54–61, 2017]. While they were able to rule out 5 of the 23 groups as potential counterexamples to GSC, they reported that their computational approach does not scale to genera higher than 2. We extend the work of Kangaslampi and Vdovina by developing two new approaches to analyzing the subgroups arising from periodic apartments in the 23 groups utilizing different combinations of SAT solving and orderly generation. We develop novel SAT encodings and a specialized orderly algorithm for the approaches, and perform an exhaustive analysis (over the 23 groups) of the genus 3 subgroups arising from periodic apartments. With the aid of massively parallel computation we also exhaust the case of genus 4. As a result we rule out 4 additional groups as counterexamples to GSC leaving 14 of the 23 groups for further inspection. In addition to this our approach provides an independent verification of the genus 2 results reported by Kangaslampi and Vdovina.

Avainsanat — Nyckelord — Keywords

Geometric group theory, hyperbolic buildings, Boolean satisfiability, combinatorial generation

Säilytyspaikka — Förvaringsställe — Where deposited

Kumpula Campus Library

Muita tietoja — Övriga uppgifter — Additional information

Contents

Chapter 1

Introduction

Computational methods, such as automated reasoning and combinatorial generation, have proven effective as means to provide insight into many fundamental open mathematical conjectures and questions. Automated reasoning and combinatorial generation have been used to settle (prove or disprove) and verify various conjectures [1, 2, 3, 4, 5, 6] as well as provide new insight into mathematical questions [7, 8, 9, 10, 11, 12] and construct examples of certain mathematical objects [13, 14, 15, 16, 17, 18, 19, 20]. In this thesis we utilize a combination of two methods from these categories—namely Boolean satisfiability [21] and orderly generation [22]—to investigate a fundamental unsolved problem in geometric group theory [23, 24] related to hyperbolic groups.

The notion of hyperbolic groups stems from the work of Gromov, who in his essay from 1987 outlines the concept [25]. A great deal of research has been done on hyperbolic groups since, and a number of significant properties of hyperbolic groups have been discovered [26, 27, 28, 29, 30, 31, 32, 33, 34]. In fact, most finitely generated groups are hyperbolic, since it has been shown to be highly likely that a *randomly* constructed finitely generated group is hyperbolic [25, 35]. In addition to the ubiquity of hyperbolic groups they have been proven to have important computational properties: their word, conjugacy and isomorphism problems are decidable [36, 37], unlike the corresponding problems of groups in general.

Hyperbolic groups can be constructed by different methods one of which is the theory of *buildings*. Buildings are mathematical constructions of geometric and combinatorial nature first defined by Jacques Tits to aid in the study of algebraic groups [38, 39, 40]. Intuitively, buildings are geometric realizations of groups yielding insight into groups through their geometric properties. The study of buildings has helped mathematicians both gain insight into the structure of certain groups as well as define entirely new groups, e.g., the twisted Chevalley group of type ${}^3D_4$ [41]. Originally research on buildings focused mostly on so-called *spherical* and *Euclidean* buildings that exhibit spherical and Euclidean geometries, respectively. Hyperbolic buildings, however, have not been studied as extensively, although interest in them has risen in recent years [42, 43, 44, 45, 46].

The problem we tackle in this thesis is the Gromov subgroup conjecture (GSC) which states that "*every one-ended hyperbolic group contains a subgroup isomorphic to the fundamental group of a surface of genus at least 2*". GSC has received a fair amount of attention in terms of classical mathematical treatment [47, 48, 49, 50, 51, 52, 53, 54, 55] as well as recently from a computational angle [56]. The conjecture has been established to hold for various hyperbolic groups [50, 48, 47, 53, 49], and it is even known that a randomly chosen one-ended hyperbolic group almost always contains a surface subgroup [54].

A class of groups for which GSC remains open consists of so-called *non-right-angled* hyperbolic groups, which hence may still provide counterexamples disproving GSC. Non-right-angled hyperbolic groups have, in fact, been constructed and studied with GSC in mind. In [57] Kangaslampi and Vdovina constructed 23 non-right-angled hyperbolic groups through the use of hyperbolic buildings.

In [58] Vdovina outlines the *polygonal construction method* for the construction of hyperbolic buildings, which is based on the work of Ballman and Brin [59, 60] as well as Gaboriau and Paulin [61]. The polygonal construction method allows for the construction of hyperbolic buildings as *universal covers* of finite polyhedra. Following this work, Kangaslampi, Vdovina, and Carbone constructed and classified examples of hyperbolic buildings using the polygonal construction method. In [57] Kangaslampi and Vdovina classify the torsion-free groups acting simply transitively on the corresponding buildings, and in [62] Carbone, Kangaslampi and Vdovina classify the corresponding torsion groups.

Motivated by Gromov's subgroup conjecture, Kangaslampi and Vdovina continued investigating the 23 torsion-free groups constructed in [57]. They used computational methods to check whether the 23 groups contain surface subgroups of genus 2 arising from so-called *periodic apartments* [56]. They first show how the existence of periodic apartments, and thus surface subgroups, reduces to a graph search problem, specifically to the existence of bipartite, 3-regular, connected graphs that decompose into 8-cycles and admit a specialized "coloring" by the representation of the group.

The size of these graphs is dictated by the parameter genus, which we denote by g. Using depth-first search, Kangaslampi and Vdovina exhaustively analyze the $g = 2$ case and discover periodic apartments in five of the 23 groups thus ruling them out as possible counterexamples to GSC. They also report that their procedure does not scale to $g = 3$ due to the sheer number of graphs that should be considered. While for $g = 2$ there are 773 bipartite, 3-regular, connected multigraphs, already for $g = 3$ the number is $\approx 13 \cdot 10^9$.

In this thesis we continue on the work of Kangaslampi and Vdovina by developing novel approaches to the graph search problem utilizing different combinations of Boolean satisfiability and orderly generation which enables scaling the results to higher genera.

Boolean satisfiability [21] refers to the satisfiability problem of propositional logic, i.e., the problem of determining whether there exists a truth assignment satisfying the input propositional formula ϕ. This problem—often referred to as the SAT problem—is an archetypal NP-complete problem [63]. Essentially Boolean satisfiability is a constraint satisfaction problem: propositional logic is the language in which the constraints are expressed with the search space being the set of truth assignments.

Although the study of Boolean satisfiability was originally of theoretical interest it has expanded into the practical domain due to the development of efficient algorithms. Specifically, the SAT problem is well suited for use in the so-called *model & solve* paradigm of declarative programming. There are numerous other NP-complete problems besides SAT, but it is the astounding performance of modern SAT solvers [64] and the flexibility of modelling with propositional logic [65, 66, 67, 68, 69] which have resulted in its widespread use. SAT solving has, in fact, been succesfully applied to settle various types of mathematical conjectures [7, 13, 14, 1, 2, 15, 8, 3, 4, 5, 16].

Besides SAT we employ orderly generation [22, 70, 71, 72, 73, 74] which is a highly versatile approach used for the generation of isomorph-free collections of combinatorial objects such as graphs. The efficiency of orderly generation stems from the fact that

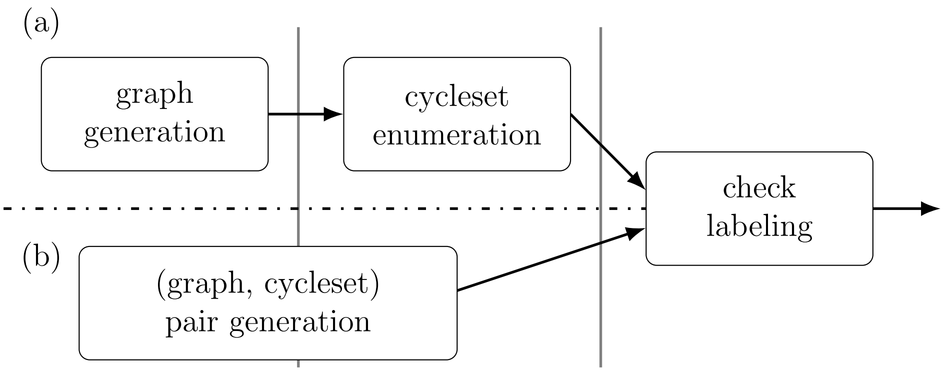


Figure 1.1: Overviews of the approaches taken in this thesis: part (a) of the diagram depicts the workflow of the direct approach whereas part (b) depicts the workflow of the orderly approach.

generating the objects in a specific order and rejecting non-canonical ones at an early stage allows for avoiding the explicit canonization of the resulting connection. Orderly generation has been succesfully applied to generate a large variety of combinatorial objects many of which have been used to study various conjectures [17, 18, 9, 6, 10, 11, 12, 71, 73, 19, 70, 72, 20, 74, 75].

To find the genera 3 and 4 periodic apartments and the corresponding surface subgroups in the 23 groups constructed in [57] we modularize the graph search problem arising from [56] into three subproblems which we solve using Boolean satisfiability and orderly generation. Particularly, we decompose the search problem into

(i) generating connected, bipartite, and 3-regular graphs of specific size,

(ii) for each of the graphs from (i), determining whether the graph admits a directed decomposition into a set of cycles of length 8, and if it does, enumerating all of such cyclesets, and

(iii) for each of the graphs admitting a cycleset decomposition from (ii), checking whether the graph oriented by its cyclesets admits a specific type of a labeling.

We develop two different approaches, the so-called *direct* and *orderly* approaches, for solving (i)–(iii) in various combinations of SAT and orderly generation as illustrated in Figure 1.1.

The direct approach consists of employing an off-the-shelf orderly generation tool called Multigraph [76] to solve part (i), and utilizing SAT for parts (ii) and (iii). For part (ii) we develop a SAT encoding representing valid cycleset decompositions of graphs from (i), which allows for the enumeration of the cycleset decompositions of each graph. For part (iii) we develop a SAT encoding modeling a valid labeling of a graph from (ii) by a group from [57], which allows for checking the existence of such a labeling.

The orderly approach depicted in Figure 1.1b first generates directly the graphs that admit a cycleset decomposition after which their cycleset decompositions are enumerated. We essentially solve (i) and (ii) by developing a specialized orderly algorithm. The remainder of the orderly approach is the same as the direct approach, namely checking each generated graph for a valid labeling by each of the 23 groups [57].

Using this combination of Boolean satisfiability and orderly generation we are able to exhaustively treat the genus 3 case as well as the genus 4 case with the aid of massive parallelization. As a result we rule out further 4 groups from the remaining set of possible counterexamples to GSC thus leaving 14 groups out of the 23 for further inspection. Our results also serve as an independent validation of the results presented in [56] for $g = 2$. The results of this thesis have been published in the Proceedings of LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning [77].

Rest of this thesis is organized as follows. We begin in Chapter 2 by explaining how the graph search problem arises from the geometric setting, and formalizing the problem decomposition. In the latter part of Chapter 2 we present several useful properties of the graphs that aid in developing efficient SAT encodings and an efficient orderly algorithm. In Chapter 3 we formulate two SAT encodings after an overview of propositional logic and SAT solving. We formulate one encoding for the extraction of cycleset decompositions of graphs and another for checking a graph for a valid labeling using groups from [57]. Chapter 4 focuses on orderly generation, beginning with a quick overview after which we develop a specialized algorithm for the generation of graphs along with their admissible cycleset decompositions. In Chapter 5 we detail the experimental setup and results achieved, and discuss both performance and correctness of the results. Chapter 6 then concludes the thesis and considers several possibilities for future work and connections to other problems. Appendix A provides the representations of groups constructed in [56], and Appendix B lists example graphs for each of the groups we rule out as possible counterexamples to GSC.

Chapter 2

Finding Surface Subgroups via Graph Search

In the first half of this chapter we formalize the problem of finding surface subgroups arising from periodic apartments in the 23 groups constructed in [57] via its reduction to a graph search problem. In the second half we prove many structural properties of the graphs that are useful in optimizing the search. We begin with necessary graph-theoretical definitions.

A graph is a pair (V, E) where V is a set of nodes and E a multiset of edges $\{v, w\}$ with $v, w \in V$. Note that it is crucial for all edges to be identifiable in the encodings we develop in Chapter 3, i.e., the encodings need to be able to tell even parallel edges apart. For the simplicity of presentation we use the multiset definition of a graph and make note of cases when we need to be able to identify parallel edges from each other.

A graph $G = (V, E)$ is simple if E contains at most one copy of each element whereas G is non-simple otherwise. Note that our definition of a graph encompasses both simple and non-simple graphs. The cardinalities of V and E are called the order and size of G, respectively. A subgraph of $G = (V, E)$ is a graph $G' = (V', E')$ for which $V' \subseteq V$ and $E' \subseteq E$. Nodes $v, w \in V$ are said to be adjacent if there exists an edge $e \in E$ containing both, and in this case node v and w are said to be incident to e. The degree of a node $v \in V$ is the number of edges $e \in E$ containing v. A graph is k-regular if all its nodes have degree k for some $k \in \mathbb{N}$. Graph $G = (V, E)$ is disconnected if its node set V can be partitioned into two sets such that there is no edge between the sets, i.e., if there are $V_1, V_2 \subseteq V$ such that $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and for all $e = \{v, u\} \in E$ either $v, u \in V_1$ or $v, u \in V_2$. A graph is connected if it is not disconnected. Graph $G = (V, E)$ is k-colorable if each of its nodes can be assigned one of k colors in such a way that no edge $e \in E$ joins two nodes of the same color. A 2-colorable graph is also referred to as being bipartite.

2.1 Setting Up the Graph Search Problem

We begin the setup of the problem focused on in this thesis by considering the 23 hyperbolic groups identified and studied by Kangaslampi and Vdovina [57], i.e., the groups whose surface subgroups we are interested in. Following the notation of [57] we denote the 23 groups by $T_1, \ldots, T_{23}$. Each of the 23 groups is finitely represented using 15 generators $x_1, \ldots, x_{15}$ and 15 relations of length 3 represented using *triplets* of the form (x_i, x_j, x_k), with the meaning $x_i x_j x_k = 1$. Note that relations are equivalent to all its

Table 2.1: The relations $x_i x_j x_k = 1$ of groups T_{15}, T_{16}, T_{17}, and T_{18}, represented as triplets (x_i, x_j, x_k) where the generators are $x_1, \ldots, x_{15}$.

T_{15}	T_{16}	T_{17}	T_{18}
(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)
(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)
(x_{11}, x_5, x_2)	(x_{11}, x_5, x_2)	(x_{11}, x_4, x_2)	(x_{11}, x_4, x_2)
(x_{14}, x_4, x_2)	(x_{14}, x_3, x_2)	(x_{14}, x_6, x_2)	(x_{14}, x_3, x_2)
(x_3, x_6, x_3)	(x_8, x_4, x_3)	(x_3, x_{12}, x_3)	(x_9, x_5, x_3)
(x_{15}, x_{12}, x_3)	(x_{14}, x_9, x_3)	(x_8, x_5, x_3)	(x_{13}, x_7, x_3)
(x_7, x_8, x_4)	(x_6, x_6, x_4)	(x_8, x_{13}, x_4)	(x_8, x_6, x_4)
(x_{15}, x_{13}, x_4)	(x_{15}, x_{13}, x_4)	(x_{14}, x_{14}, x_4)	(x_{14}, x_8, x_4)
(x_8, x_7, x_5)	(x_7, x_7, x_5)	(x_9, x_7, x_5)	(x_6, x_{12}, x_5)
(x_{14}, x_9, x_5)	(x_{15}, x_{12}, x_5)	(x_{11}, x_9, x_5)	(x_{15}, x_{13}, x_5)
(x_9, x_{11}, x_6)	(x_{14}, x_{11}, x_6)	(x_7, x_8, x_6)	(x_7, x_9, x_6)
(x_{11}, x_{13}, x_6)	(x_{11}, x_{13}, x_7)	(x_{15}, x_{12}, x_6)	(x_{11}, x_{10}, x_7)
(x_{10}, x_9, x_7)	(x_9, x_{12}, x_8)	(x_{10}, x_{13}, x_7)	(x_{14}, x_{12}, x_8)
(x_{12}, x_{12}, x_8)	(x_{10}, x_9, x_8)	(x_{11}, x_{10}, x_9)	(x_{13}, x_{11}, x_9)
(x_{13}, x_{14}, x_{10})	(x_{13}, x_{12}, x_{10})	(x_{15}, x_{13}, x_{12})	(x_{15}, x_{12}, x_{10})

cyclic permutations which can be shown by straightforward algebraic manipulation. The relation $x_i x_j x_k = 1$, for example, is equivalent to $x_j x_k x_i = 1$ as well as $x_k x_i x_j = 1$. As examples of these groups, the relations of T_{15}, T_{16}, T_{17} and T_{18} are listed in Table 2.1. A complete listing of the representations of each of the 23 groups is provided in Appendix A. Observe that the relations may contain multiple instances of the same generator. The group T_{15}, for example, contains the triplet (x_1, x_{15}, x_1) which has two occurrences of generator x_1.

These groups are examples of *non-right-angled* hyperbolic groups, a class of groups for which Gromov's subgroup conjecture remains open. In [56] Kangaslampi and Vdovina presented a computational study of these 23 groups they constructed earlier [57]. Specifically, they showed in [56] that the question of whether a particular one of these groups contains a *surface subgroup*, i.e., a subgroup isomorphic to the fundamental group of a closed surface, arising from so-called *periodic apartments* is equivalent to determining whether a specific type of a graph (with a non-trivial combination of properties) exists. In other words, the existence of a periodic apartment implies the existence of a surface subgroup, which in turn implies that the particular group in question is provably not a counterexample to Gromov's subgroup conjecture.

Particularly, the graphs whose existence implies the existence of a periodic apartment have the following properties.

(i) The graphs are bipartite, connected and 3-regular, and their order and size depend on the parameter g (genus).

(ii) The graphs admit a directed decomposition into cycles of length 8.

(iii) The graphs admit a simultaneous labeling of vertices and edges by the representation of the group in question subject to specific constraints.

In the following we formulate three parameterized sets of graphs $\mathsf{base}(g)$, $\mathsf{cycles}(g)$ and $\mathsf{labels}(T_i, g)$ corresponding to (i), (ii) and (iii), respectively. Before diving into the details

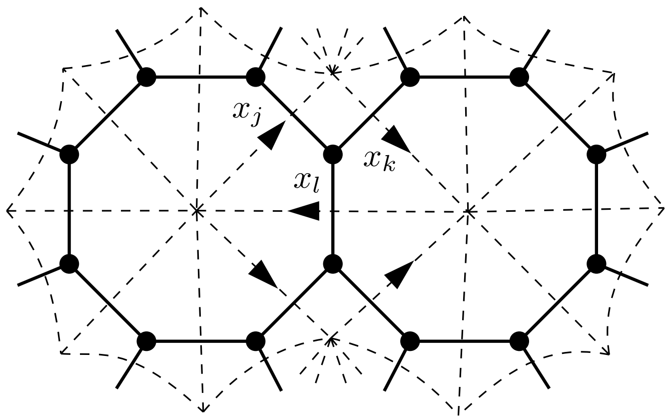


Figure 2.1: Tesselation with hyperbolic triangles whose angles are $\frac{\pi}{4}$.

and elaborating the constraints formally, we briefly explain how these constraints arise from the problem of finding a periodic apartment in one of the 23 groups $\{T_1, \ldots, T_{23}\}$. For a complete description of how exactly the constraints for the graphs arise we refer the reader to [56].

Let $g > 1$ be a fixed natural number and T_i one of the 23 groups. Each relation (x_j, x_k, x_l) of T_i determines an oriented triangle whose edges are labeled with x_j, x_k and x_l. A periodic apartment of genus g is then a surface of genus g ("donut with g holes") constructed from these triangles in a way that the labels and orientations match. The graph we wish to find is the dual graph of this triangulation, i.e., a graph whose nodes represent the triangles with an edge between two nodes if the respective triangles share an edge, see Figure 2.1 for an illustration.

Observe that the graphs must be 3-regular since they represent a triangulation, and bipartite because every other triangle must have "opposite" orientation. If all triangles had the same orientation their edges could not be matched to build the surface. Now the triangles we consider here are assumed to be hyperbolic with all angles $\frac{\pi}{4}$. From this it then follows that exactly 8 triangles intersect at every corner. The number of nodes and edges can be deduced to be $16(g-1)$ and $24(g-1)$ using Euler's formula $V - E + F = 2 - 2g$ as follows [56]. We can consider the surface to be tesselated by regular octagons due to the triangles having angles $\frac{\pi}{4}$ (See Figure 2.1). Now each vertex meets 3 octagons whereas each edge meets 2. Denoting the number of octagons by F we thus deduce that $V = \frac{8F}{3}$ and $E = \frac{8F}{2} = 4F$ which combined with Euler's formula yields $F = 6(g-1)$. The number of 8-cycles is thus $6(g-1)$ whereas the order and size of the dual graph are $16(g-1)$ and $24(g-1)$, respectively.

Definition 1. *Let $G = (V, E)$ be a graph and $g > 1$ be a natural number. Then $G \in \mathsf{base}(g)$ if $|V| = 16(g-1)$, $|E| = 24(g-1)$, and G is connected, bipartite, and 3-regular.*

Observe that, due to 3-regularity and connectedness of graphs in $\mathsf{base}(g)$, two nodes can have at most two edges between them. These pairs of parallel edges are called *double edges*.

Next we consider the graphs in $G = (V, E) \in \mathsf{base}(g)$ which admit a *cycleset*. Let $\mathsf{directed}(E) = \{(v_1, v_2), (v_2, v_1) \mid \{v_1, v_2\} \in E\}$ denote the decomposition of E into directed edges. A cycleset for $G = (V, E) \in \mathsf{base}(g)$ is a set of $6(g-1)$ cycles of length 8 in $\mathsf{directed}(E)$ covering the directed decomposition[1]. Observe that each di-

[1]We follow the notation in [56] and refer to these structures as cycles instead of walks.

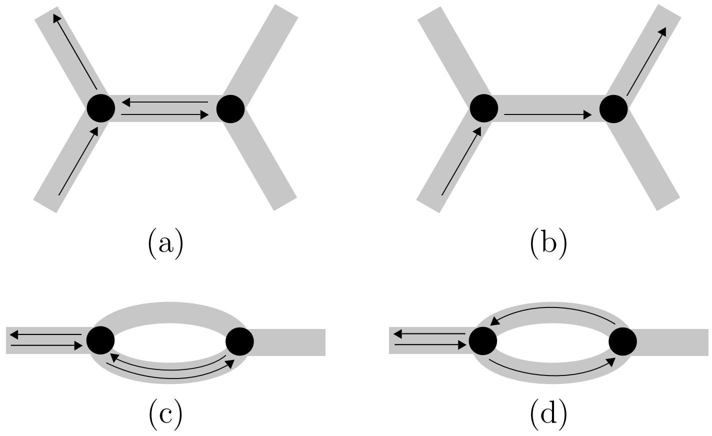


Figure 2.2: Two invalid ways of routing 8-cycles are shown in (a) and (c). Subfigures (b) and (d), on the other hand, show two valid ways of routing 8-cycles.

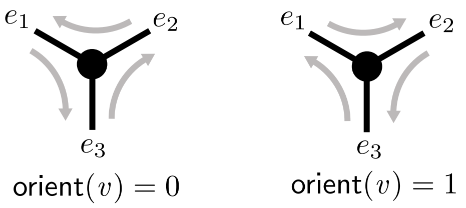


Figure 2.3: The 2 possible orientations of a node.

rected edge is covered exactly once by the decomposition: The number of directed edges $|\mathsf{directed}(E)| = 2|E| = 48(g-1)$ equals exactly the number of edges required $8 \cdot 6(g-1) = 48(g-1)$. This implies that each undirected edge is traversed twice (once in each direction).

Formally, an 8-cycle in G is a sequence $(e_0, \ldots, e_7)$, where for each $i \in \{0, \ldots, 7\}$, $e_i \in \mathsf{directed}(E)$ and for the end-points of consecutive edges $e_i = (v', v)$ and $e_{(i+1) \bmod 8} = (v, v'')$ it holds that $v' \neq v''$ if e_i and $e_{(i+1) \bmod 8}$ originate from the same undirected edge. Special care needs to be taken with this definition in the case of non-simple graphs as noted in [56]. The definition forbids an 8-cycle from "doubling back", i.e., the 8-cycle cannot contain edges $e = (a, b)$ and $e' = (b, a)$ in subsequent positions if e and e' arise from the same undirected edge. If the directed edges e and e', however, correspond to different undirected edges we may have an 8-cycle containing e and e' at consecutive positions, see Figure 2.2 for an illustration.

Definition 2. *Let $G = (V, E)$ be a graph such that $G \in \mathsf{base}(g)$ for some $g > 1$. Then $G \in \mathsf{cycles}(g)$ if G contains a cycleset, i.e., a set of $6(g-1)$ 8-cycles, where each edge in $\mathsf{directed}(E)$ is traversed exactly once.*

There may be several cyclesets for $G \in \mathsf{cycles}(g)$; we denote by $\mathsf{cyclesets}(G)$ the set of all cyclesets of G. Hence $\mathsf{cyclesets}(G) = \emptyset$ implies $G \notin \mathsf{cycles}(g)$ and vice versa. Observe that a cycleset covers all the edges in G, and this allows one to uniquely order the incident edges of each node. There are exactly two ways in which the cycles can pass through a node (see Figure 2.3), and hence each node has an *orientation* determined by the cycles passing through it. Let $\mathsf{edges}(v)$ be the set of edges incident to vertex v for each $v \in V$. The orientation of a node v is represented using an *order function* O_v mapping each $e \in \mathsf{edges}(v)$ to its successor. For instance, in the topmost case in Figure 2.3 the order function O_v is defined as $O_v(e_1) = e_3$, $O_v(e_2) = e_1$, and $O_v(e_3) = e_2$.

Table 2.2: A cycleset of graph G_7^2.

1	3	2	1	4	5	6	7
2	14	15	12	16	17	14	3
4	7	8	9	10	11	12	13
5	13	15	17	18	19	10	20
6	20	9	21	22	23	21	8
11	19	24	22	23	24	18	16

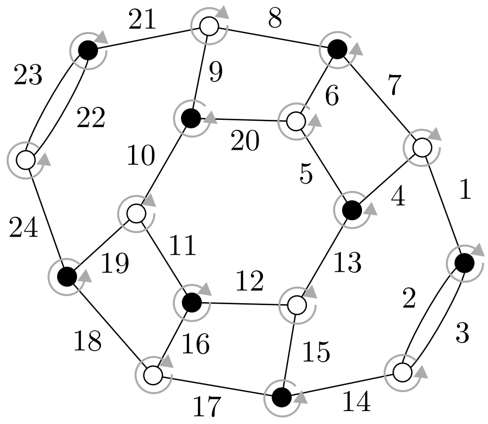


Figure 2.4: Graph G_7^2

Example 1. *Consider the graph G_7^2 shown in Figure 2.4. Observe that G_7^2 is a non-simple graph with two double edges. Now, $G_7^2 \in$ base(2), i.e., G_7^2 is bipartite, connected, 3-regular and has 16 nodes and 24 edges. The nodes of the graph are colored black/white to indicate bipartiteness.*

Furthermore, $G_7^2 \in$ cycles(2), i.e., cyclesets(G_7^2) $\neq \emptyset$. One of the cyclesets of G_7^2 is listed in Table 2.2 (using the names of undirected edges to avoid obscuring the figure too much). Each node is passed through exactly three times by the cycles (in the case of double edges the node is passed twice by the same cycle and once by another). Each undirected edge is traversed twice (once in each direction). The orientations arising from the cycleset in Table 2.2 are denoted using arrows around the nodes in Figure 2.4.

We are now ready to define the graphs that represent periodic apartments of the hyperbolic building corresponding to some group T_i. Recall that the nodes of any such graph represent triangles whose sides have labels x_i and edges between nodes represent the adjacency of the triangles. To represent the action of group T_i on the apartment, we need to define conditions for a *valid labeling* for $G \in$ cycles(g), which corresponds to a labeling of the sides of the triangles.

Definition 3. *Let $G = (V, E) \in \mathsf{cycles}(g)$ for some $g > 1$. A labeling of G using group T_i, denoted by (L_v, L_e), consists of two functions:*

- *L_v mapping $v \in V$ to relations (x_i, x_j, x_k) of T_i, and*
- *L_e mapping $e \in E$ to generators x_i of T_i,*

in a way that the label of node v matches the labels of $e \in \mathsf{edges}(v)$, i.e., for all $v \in V$ it holds that if $L_v(v) = (x_i, x_j, x_k)$, then $\{L_e(e) \mid e \in \mathsf{edges}(v)\} = \{x_i, x_j, x_k\}$.

The acceptability of a labeling depends on the orientations of the nodes of G. The orientations are determined by the cyclesets of $G \in \mathsf{cycles}(G)$ together with a chosen 2-coloring (due to bipartiteness). We assume here a *fixed* 2-coloring of G using colors black and white. Intuitively, the labeling of a *white node* has to *match the orientation* of the node, and the labeling of a *black node* has to *match the inverted orientation* of the node. Furthermore, two adjacent nodes in G cannot be labeled with the same triplet unless (1) the triplet contains two occurrences of the same generator, (2) the connecting edge is labeled with the duplicated generator, and (3) the index of the generator in the triplet is different for both nodes. For a detailed discussion on how exactly these constraints arise, we refer the reader to [56]. For an intuition of the conditions (1)–(3), recall that the triangles tesselating the "surface with g holes" are oriented and have their sides labeled with elements x_i (generators of T_i), and the sides of two adjacent triangles that overlap must have the same label. Additionally, due to the nature of the group action, two triangles of the *same type* (i.e., labelled using the same triplet of T_i) cannot share the same side. This means that two triangles of the same type with three *different* labels x_i, x_j and x_k cannot be adjacent. Two triangles of the same type, however, can be adjacent if their labels are x_i, x_i and x_j, since then they may be attached from different sides which have the same label, see Figure 2.5.

Definition 4. *Let $G = (V, E) \in \mathsf{cycles}(g)$ for some $g > 1$ and $W \in \mathsf{cyclesets}(G)$. A labeling (L_v, L_e) of G using T_i respects the orientation induced by W if the following conditions hold for each $v \in V$ with $\mathsf{edges}(v) = \{e_1, e_2, e_3\}$ and orientation $O_v(e_1) = e_2$, $O_v(e_2) = e_3$, $O_v(e_3) = e_1$ in W.*

(i) $L_v(v) = (L_e(e_1), L_e(e_2), L_e(e_3))$ if v is white.

(ii) $L_v(v) = (L_e(e_3), L_e(e_2), L_e(e_1))$ if v is black.

(iii) For each $e = \{v, w\} \in E$, if $L_v(v) = L_v(w)$, then this label (triplet) has two occurrences of the same generator x_i, $L_e(e) = x_i$, and the orientations of v and w are such that e has different position (index) in $L_v(v)$ and $L_v(w)$.

Given $W \in \mathsf{cyclesets}(G)$, a labeling using T_i is *valid* with respect to W if it satisfies conditions (i)–(iii) of Definition 4, and otherwise *invalid*.

Example 2. *Figure 2.6 illustrates examples of valid and invalid labelings. In (a)–(c), valid labelings in the neighborhood of a white node, a black node, and for two adjacent nodes which are assigned the same triple, respectively, are illustrated. Invalid labels are illustrated in (d)–(f). In (d) the labels of the edges do not match the triple; in (e) the labels match the triplet but in the wrong order; and (f) illustrates how a labeling of two adjacent nodes which are assigned the same triplet may fail. Here the generators x_1 in bold in (c) and (f) denote the elements of the triplets corresponding to the label of the connecting edge.*

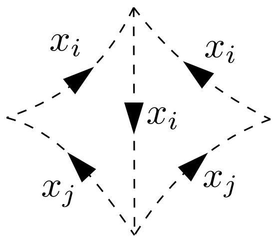


Figure 2.5: Invalid (left) and valid (right) adjacent triangles.

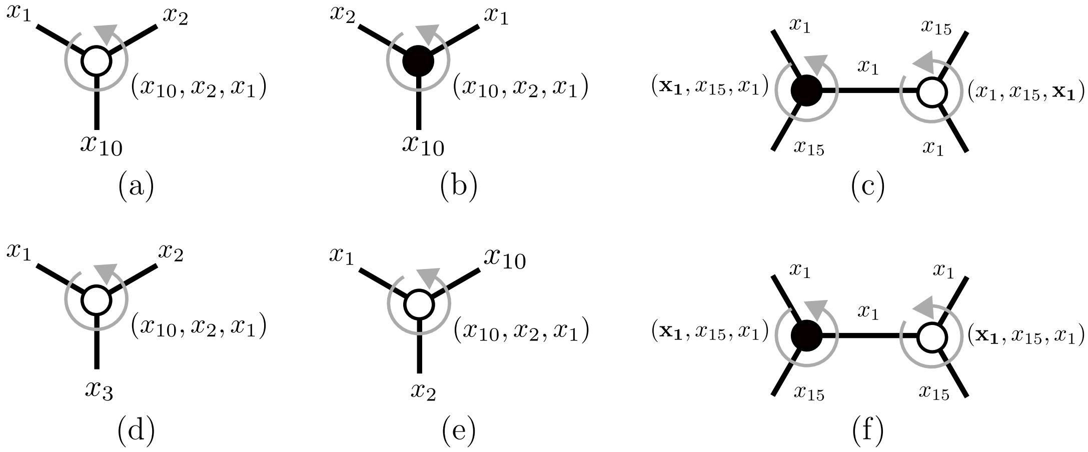


Figure 2.6: Examples of valid labelings in (a)–(c) and invalid labelings (d)–(f), see Example 2.

Finally, we define the set of graphs that admit a valid labeling.

Definition 5. *Let $G = (V, E)$ be a graph such that $G \in \mathsf{cycles}(g)$ for some $g > 1$, and let T_i be one of the 23 groups constructed in [57]. We say that $G \in \mathsf{labels}(T_i, g)$, if for some set $W \in \mathsf{cyclesets}(G)$ there exists a valid labeling (L_v, L_e) of G using T_i with respect to W.*

Observe that $\mathsf{labels}(T_i, g) \subseteq \mathsf{cycles}(g) \subseteq \mathsf{base}(g)$ for all $g > 1$ and T_i such that $i \in \{1, \ldots, 23\}$. The existence of a graph $G \in \mathsf{labels}(T_i, g)$ is connected to the existence of surface subgroups in T_i as follows.

Theorem 1 ([56])**.** *Let T_i be one of the 23 groups constructed in [57] and $g > 1$ a natural number. If $\mathsf{labels}(T_i, g) \neq \emptyset$, then there exists a periodic apartment in the hyperbolic building corresponding to T_i that is invariant under the action of a genus g surface.*

The existence of a periodic apartment in the hyperbolic building implies the existence of a surface subgroup of genus g. Hence, if $\mathsf{labels}(T_i, g) \neq \emptyset$, then Gromov's subgroup conjecture holds for T_i. It is not known, however, whether the existence of a surface subgroup of genus g implies the existence of a periodic apartment.

Corollary 1. *Let T_i and g be as defined in Theorem 1. If $\mathsf{labels}(T_i, g) \neq \emptyset$, then there exists a subgroup in T_i isomorphic to the fundamental group of a genus g surface.*

2.2 Properties of Graphs in the Search Space

In this section we prove many useful structural properties satisfied by graphs in $\mathsf{cycles}(g)$ for $g > 1$. The properties we show arise mainly from the interaction between double

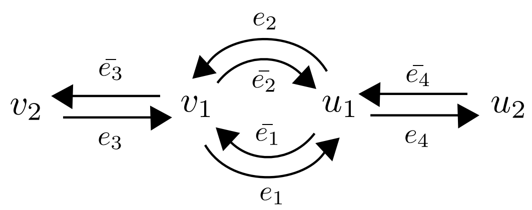
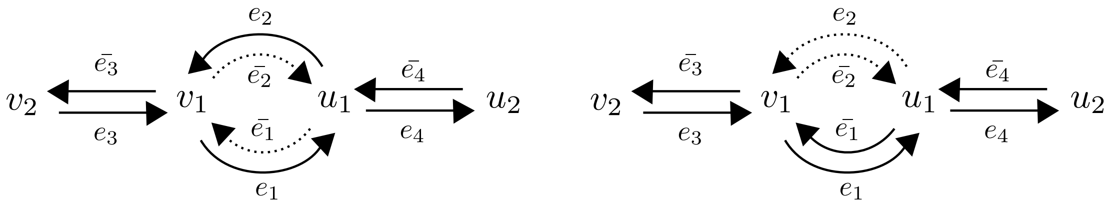


Figure 2.7: Directed decomposition of a double edge.



(a) (b)

Figure 2.8: Unacceptable routing of cycles near double edges.

edges and 8-cycles. We also prove a connection between the cyclesets of $G \in \mathsf{cycles}(g)$ and the orientations of nodes.

First we show that the existence of a cycleset in a graph $G \in \mathsf{cycles}(g)$ implies a low upper bound on the number of double edges. To prove this we first demonstrate a result already used in [56] that shows how 8-cycles can pass through double edges. Using this observation we then demonstrate a correspondence between the numbers of cycles and double edges yielding us a useful upper bound.

For any graph $G = (V, E)$ and each $e \in \mathsf{directed}(E)$ we use $\bar{e}$ to denote the opposite directed edge originating from the same undirected edge in E. In other words if $e = (a, b) \in \mathsf{directed}(E)$ then $\bar{e} = (b, a)$, and the pair $(e, \bar{e})$ is the result of splitting an undirected edge $e^* = \{a, b\} \in E$. In the following we refer to truncations of an 8-cycle as a *subwalk*. The sequence (e_3, e_4, e_5), for example, is a length-3 subwalk of the 8-cycle $(e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8)$.

Lemma 1 ([56]). *Let $g > 1$ be a natural number, $G = (V, E) \in \mathsf{cycles}(g)$, $W \in \mathsf{cyclesets}(G)$ and $w \in W$ a cycle passing through a double edge. Now w contains subwalk $(e_3, e_1, e_2, \bar{e}_3)$ with $e_1, e_2, e_3 \in \mathsf{directed}(E)$ arising from distinct undirected edges incident to the same nodes.*

Proof. Assume that $G = (V, E) \in \mathsf{cycles}(g)$ for some $g > 1$, $v_1, v_2, u_1, u_2 \in V$ and $e_i, \bar{e_i} \in \mathsf{directed}(E)$ for $i \in \{1, 2, 3, 4\}$ as depicted in Figure 2.7. Let w be the cycle in $W \in \mathsf{cyclesets}(G)$ containing edge e_3. The successor of e_3 in w must be either e_1 or $\bar{e_2}$ with both choices leading to a symmetrical situation. We assume that e_1 follows e_3 in w, i.e., w contains subwalk (e_3, e_1). Now edge e_1 in w must be followed by either e_2 or e_4, with the first choice necessarily leading to w containing subwalk $(e_3, e_1, e_2, \bar{e_3})$ and the second choice leading to a contradiction as we will demonstrate next.

We now assume that w contains (e_3, e_1, e_4). Since the cycles in W cover the entire graph, we know that some cycle $w' \in W$ contains $\bar{e_3}$. The cycle w' now contains either $(\bar{e_4}, e_2, \bar{e_3})$ or $(\bar{e_4}, \bar{e_1}, \bar{e_3})$. In the first case edges $\bar{e_1}$ and $\bar{e_2}$ would be left orphaned and unable to participate in any 8-cycle as shown in Figure 2.8a. The second case, shown in Figure 2.8b, would result in edges e_2 and $\bar{e_2}$ being orphaned. This is a contradiction since W was assumed to be a cycleset, which by definition covers the entire graph. $\square$

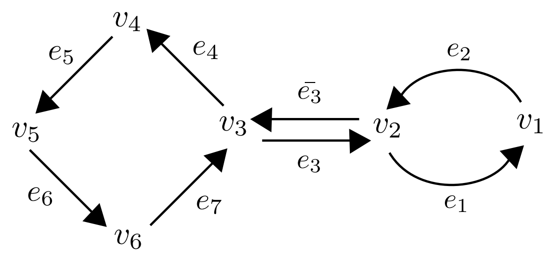


Figure 2.9: An 8-cycle passing a double edge.

We then show that each 8-cycle may traverse at most one double edge.

Theorem 2. *Let $g > 1$ be a natural number, $G = (V, E) \in \mathsf{cycles}(g)$, $W \in \mathsf{cyclesets}(G)$ and $w \in W$ a cycle. Now w contains at most one subwalk of the form $(e_3, e_1, e_2, \bar{e_3})$ where e_1, e_2 and e_3 arise from distinct undirected edges.*

Proof. Assume to the contrary that cycle w contains two subwalks of the form $(e_3, e_1, e_2, \bar{e_3})$ as stated in the theorem. We denote $w = (e_3, e_1, e_2, \bar{e_3}, e_4, e_5, e_6, e_7)$, see Figure 2.9 for an illustration. Now cycle w must contain directed edges x and $\bar{x}$ arising from the same undirected edge and additionally $\bar{x}$ must be the third edge following x in w since the subwalk is of the form $(x, *, *, \bar{x})$. Note first that x cannot be e_3 or $\bar{e_3}$ since each directed edge may be used only once. The possible pairs $(x, \bar{x})$ are thus (e_1, e_4), (e_2, e_5), (e_4, e_7), (e_6, e_1) and (e_7, e_2).

If $(x, \bar{x}) = (e_1, e_4)$ we would have that $v_1 = v_3$ and $v_2 = v_4$, which would imply that v_1 and v_2 have degree greater than 3. If $(x, \bar{x}) = (e_2, e_5)$, on the other hand, we would have that $v_1 = v_5$ and $v_2 = v_4$ and again v_2 would have degree at least 4. The cases (e_6, e_1) and (e_7, e_2) are handled similarly.

The case $(x, \bar{x}) = (e_4, e_7)$, however, fails for an entirely different reason. Assuming $(x, \bar{x}) = (e_4, e_7)$ we know, since the graph is 3-regular, that there is a third undirected edge incident to v_3 whose directed components we denote by a and $\bar{a}$, see Figure 2.10. Edges e_3, $\bar{e_3}$, e_4 and $\bar{e_4} = e_7$ cannot be a part of any other cycle. Now, since all edges are used by the cycles, there must exist a cycle $w' \in W$ containing subwalk $(a, \bar{a})$. But this is a contradiction since no cycle may contain $(e, \bar{e})$ for any $e \in \mathsf{directed}(E)$. □

This yields the corollary that each double edge has to meet two distinct cycles.

Corollary 2. *Each double edge is traversed by two distinct cycles.*

Proof. Consider the directed decomposition of a double edge as depicted in Figure 2.7. The only way to use up all directed edges is if there are cycles $w, w' \in W$ such that w contains $(e_3, e_1, e_2, \bar{e_3})$ and w' contains $(\bar{e_4}, \bar{e_1}, \bar{e_2}, e_4)$. Additionally w and w' must be distinct due to Theorem 2 since each 8-cycle may contain only one subwalk of type $(x, y, z, \bar{x})$. □

Using the two previous results we finally show an upper bound for the number of double edges.

Theorem 3. *Let $g > 1$ be a natural number and $G \in \mathsf{cycles}(g)$. There are at most $3(g-1)$ double edges in G.*

Proof. Definition 2 states that graph $G \in \mathsf{cycles}(g)$ decomposes into $6(g-1)$ cycles of length 8. Taken together, Theorem 2 and Corollary 2 state that each double edge is traversed by two distinct cycles, neither of which meet any other double edges. The maximum number of double edges in $G \in \mathsf{cycles}(g)$ is thus $6(g-1)/2 = 3(g-1)$. □

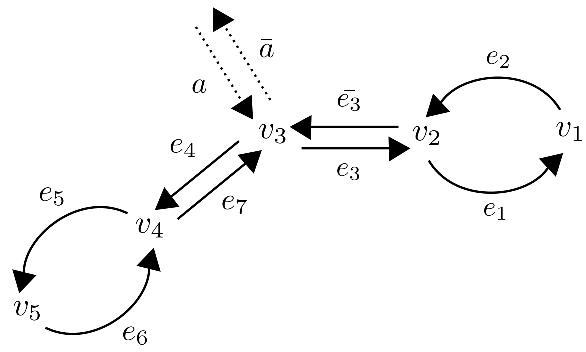


Figure 2.10: An impossible 8-cycle.

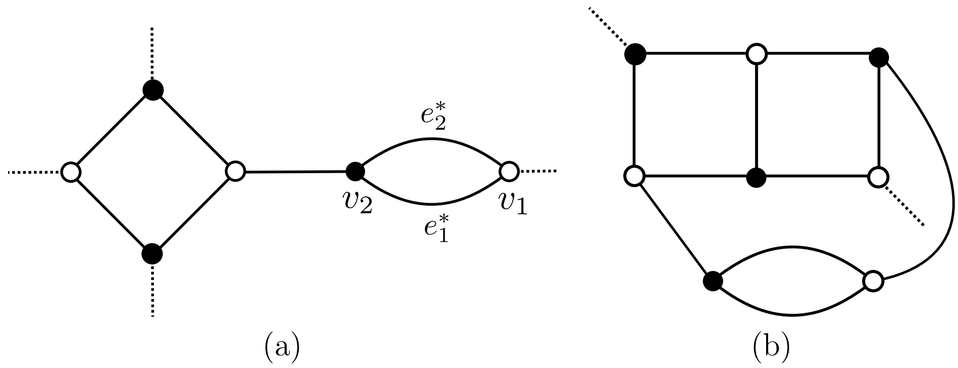


Figure 2.11: (a) Square gadget connected to node v_2. (b) Attached square gadgets.

The upper bound on the number of double edges in $G \in \mathsf{cycles}(g)$ is remarkably low and offers opportunities for optimizing the generation and search of graphs admitting cyclesets, i.e., $G \in \mathsf{cycles}(g)$. A much less remarkable upper bound of $8(g-1)$ double edges is deducible for graphs in $\mathsf{base}(g)$, and this bound for $\mathsf{base}(g)$ is in fact strict since graphs admitting $8(g-1)$ double edges exist in $\mathsf{base}(g) \setminus \mathsf{cycles}(g)$.[2] Additionally there are graphs with n double edges in $\mathsf{base}(g)$ for every $n \in \{0, \dots, 8(g-1)\}$ a significant amount of which have $n > 3(g-1)$. This indicates that starting the search procedure with input from $\mathsf{cycles}(g)$ instead of $\mathsf{base}(g)$ should result in a significant reduction of the search space.

Next we show that the existence of double edges in $G \in \mathsf{cycles}(g)$ implies the presence of a specific subgraph near each double edge in G. Using this result we are then able to give a useful lower bound on the distance between distinct double edges in G.

Theorem 4. *Let $g > 1$ be a natural number and $G = (V, E) \in \mathsf{cycles}(g)$. Assume that $v_1, v_2 \in V$ and e_1^*, e_2^* are distinct undirected edges $\{v_1, v_2\}$. Now v_2 has a square attached to it as depicted in Figure 2.11a.*

Proof. Since $G \in \mathsf{cycles}(g)$ we know that $\mathsf{cyclesets}(G) \neq \emptyset$. Let $W \in \mathsf{cyclesets}(G)$ and $w \in W$ a cycle passing v_2 twice, see Figure 2.9. To prove that v_2 indeed has an attached square subgraph in G it suffices to show that v_3, v_4, v_5 and v_6 are distinct nodes. Clearly nodes adjacent in Figure 2.9 must be distinct since the contrary would imply the existence of a loop in G. Therefore node v_3 must be distinct from v_4 and similarly for pairs (v_4, v_5), (v_5, v_6) and (v_6, v_3). Nodes v_3 and v_5, on the other hand, must be distinct since $v_3 = v_5$ would imply that $\bar{e_4} = e_5$ and further that w contains subwalk $(e_4, \bar{e_4})$.

[2] Consider for example a cycle graph with alternating double and single edges.

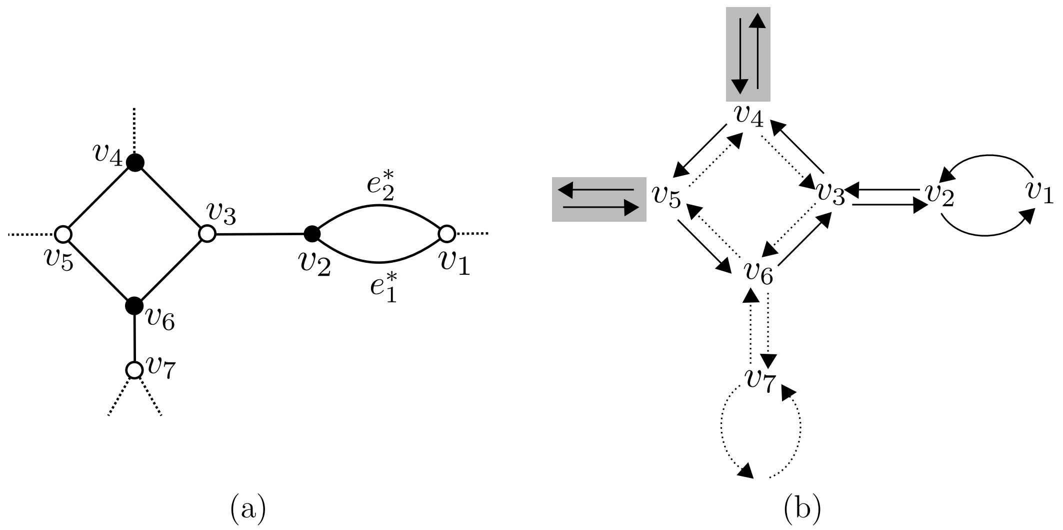


Figure 2.12: (a) Double edge with a gadget. (b) Two entangled 8-cycles.

Now assume that $v_4 = v_6$. Due to 3-regularity we know that either e_5 and e_6 or e_4 and e_7 originate from the same undirected edge. The first case implies that $\bar{e_5} = e_6$, and further that w contains $(e_5, \bar{e_5})$, a contradiction. The case $\bar{e_4} = e_7$, on the other hand, was shown to result in a contradiction in Theorem 2. Therefore v_4 and v_6 must also be distinct, and the proof is finished. □

Note that the result of the previous theorem applies to both nodes v_1 and v_2. Thus, each node of a double edge has a "square gadget" attached to it. The gadgets attached to different nodes, however, are not necessarily disjoint, see Figure 2.11b.

Corollary 3. *The minimum distance between nodes incident to double edges (excluding the neighboring node) is 4.*

Proof. The proof follows by straightforward case analysis. Assume a situation as depicted in Figure 2.12a. A double edge at a distance of 1 from v_2 would have to be at v_3, which is clearly impossible since the graph is 3-regular. A double edge at distance of 2, on the other hand, would have to be located next to v_4 or v_6, which is again impossible due to 3-regularity. Similarly node v_5, which is at distance 3 from v_2, cannot have a double edge. The only remaining node to consider is v_7, which is also at distance 3 from v_2.

We thus assume that node v_7 has a double edge. This implies that the cycle passing v_7 twice has the form depicted in Figure 2.9. Now the only way to map the 8-cycles is the one in Figure 2.12b, which leaves the remaining edges of v_4 and v_5 orphaned thereby causing some 8-cycle to contain $(x, \bar{x})$ for some $x \in \mathsf{directed}(E)$. This is a contradiction, and therefore v_7 cannot have a double edge either. Since any edge at distance < 3 from v_2 cannot have a double edge, the minimum distance from v_2 to another node with a double edge is at least 4. □

Graphs with double edges at distance 4 from each other exist implying that the bound cannot be improved. See, e.g., graph G^3_{112} on page 11 in [56].

Now we proceed to consider the correspondence of cyclesets to the orientations of nodes. Let $g > 1$ be a natural number and $G = (V, E) \in \mathsf{base}(g)$. Recall that an orientation of a node $v \in V$ is an order function O_v that maps each undirected edge to its successor. For example O_v defined by $O_v(e_1^*) = e_2^*$, $O_v(e_2^*) = e_3^*$ and $O_v(e_3^*) = e_1^*$ is an

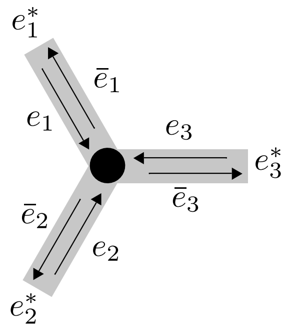


Figure 2.13: Different orientations of a node.

orientation of $v \in V$ given that the incident undirected edges of v are e_1^*, e_2^* and e_3^*, see Figure 2.13. In the following we instead represent orientations of nodes as sets of pairs of directed edges as follows. Denote the directed decomposition of the edges incident to v by e_i and $\bar{e}_i$ for $i \in \{1, 2, 3\}$. Now we may represent the orientation O_v defined above as the set $O_v^{\mathrm{dir}} = \{(e_1, \bar{e}_2), (e_2, \bar{e}_3), (e_3, \bar{e}_1)\}$. With this notation we may now state and prove that orientations of nodes split $\mathsf{directed}(E)$ into disjoint cycles.

Theorem 5. *Let $g > 1$ be a natural number, $G = (V, E) \in \mathsf{base}(g)$ and O_v^{dir} an orientation for each $v \in V$. Now $\bigcup_{v \in V} O_v^{dir}$ is a union of cycles covering all of $\mathsf{directed}(E)$.*

Proof. Let $e = (v, v') \in \mathsf{directed}(E)$ be an arbitrary directed edge. Observe first that e is (i) contained in one pair $(*, e)$ of O_v^{dir}, (ii) contained in one pair $(e, *)$ of $O_{v'}^{\mathrm{dir}}$, and (iii) not contained in any O_u^{dir} where $u \neq v, v'$.

Now the relation $\bigcup_{v \in V} O_v^{\mathrm{dir}} \subset \mathsf{directed}(E) \times \mathsf{directed}(E)$ contains exactly one pair of the form $(*, e)$ which defines a unique predecessor for each $e \in \mathsf{directed}(E)$. And similarly the unique pair of the form $(e, *)$ defines a unique successor for each $e \in \mathsf{directed}(E)$.

The partition of $\mathsf{directed}(E)$ is then extracted as the equivalence classes of the reflexive-transitive closure of $\bigcup_{v \in V} O_v^{\mathrm{dir}}$. Furthermore, each partition must represent a cycle in G since every $e \in \mathsf{directed}(E)$ has a unique predecessor and successor, i.e., no e is an end or a beginning of a path. □

In other words we just proved that the assignment of an orientation to each node of a graph $G \in \mathsf{base}(g)$ gives rise to a disjoint set of cycles in G. The lengths of the cycles, however, are not necessarily 8. A natural next question is whether every *cycleset*, i.e., set of 8-cycles, corresponds to some assignment of orientations to nodes. The following theorem demonstrates that the answer is yes, but first we need to define some notation.

Let W be a cycleset of $G = (V, E) \in \mathsf{cycles}(g)$ where $g > 1$ is a natural number. We define $\mathsf{rel}(W)$ to be the relational representation of cycleset W, i.e.,

$$\mathsf{rel}(W) = \{(e, e') \mid e' \text{ is the successor of } e \text{ in some cycle } w \in W\}.$$

In other words $\mathsf{rel}(W) \subset \mathsf{directed}(E) \times \mathsf{directed}(E)$ is the collection of predecessor-successor pairs of edges according to the cycles in W.

Theorem 6. *Let $g > 1$ be a natural number, $G = (V, E) \in \mathsf{cycles}(g)$, and $W \in \mathsf{cyclesets}(G)$. Now $\mathsf{rel}(W) = \bigcup_{v \in V} O_v^{dir}$ for some assignment of orientations O_v^{dir} for all $v \in V$.*

Proof. We prove the statement by showing that a suitable set of orientations can be directly extracted from $\mathsf{rel}(W)$. Let $v \in V$ be an arbitrary node of G and denote the directed edges pointing towards v by e_1, e_2 and e_3. Now the edges pointing away from v are $\bar{e}_1$, $\bar{e}_2$ and $\bar{e}_3$, see Figure 2.13. The possible orientations are $\{(e_1, \bar{e}_2), (e_2, \bar{e}_3), (e_3, \bar{e}_1)\}$ and $\{(e_1, \bar{e}_3), (e_3, \bar{e}_2), (e_2, \bar{e}_1)\}$. We know that exactly one of the orientations must be a subset of $\mathsf{rel}(W)$ since W covers the entire graph and these are the only possible ways to route the cycles around v. We thus define O_v^{dir} to be the orientation that is a subset of $\mathsf{rel}(W)$. Now clearly $\bigcup_{v \in V} O_v^{\text{dir}} \subseteq \mathsf{rel}(W)$ and the other direction follows from the fact that every pair $(e, e') \in \mathsf{rel}(W)$ is in some O_v^{dir}. $\square$

We have now proved in Theorems 5 and 6 that each assignment of valid orientations to nodes of $G \in \mathsf{base}(g)$ corresponds to some set of cycles covering G and that each set of 8-cycles corresponds to some assignment of orientations. This may seem simple and obvious, but it provides a powerful aid in formulating SAT encodings. This is because the orientation of a node v is a *local* property, i.e., only edges incident to v need to be referred to. The existence of a set of cycles covering a graph G, however, is a *global* property, i.e., all edges of G need to be referred to. What the previous two theorems then show is that we can enforce a global property of a graph by referring to only local properties of its nodes, and this is exactly what we do in the encoding presented in Section 3.2. Additionally, the proof of Theorem 6 yields us a procedure for extracting the orientations of nodes implied by a cycleset.

Chapter 3

Cycleset Decompositions and Group Labelings via SAT

In this chapter we develop two SAT encodings, one for enumerating the cycleset decompositions of a graph and another for checking a graph for a labeling by one of the groups from [57]. Before formulating the encodings we present an overview on Boolean modelling and SAT solving.

3.1 Boolean Satisfiability

Propositional logic is a formal language consisting of inductively defined formulas and an associated semantics for the formulas. The formulas are constructed inductively from atoms (propositional variables) $p_0, p_1, \ldots$ and the usual connectives: negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow$ and bi-implication $\leftrightarrow$. The formulas are interpreted over *assignments* which are functions mapping each propositional variable to 0 or 1. A formula ϕ is satisfied by assignment s if ϕ evaluates to 1 under the usual semantics of the connectives with each atom p_i taking on value $s(p_i)$. An assignment s satisfying formula ϕ is referred to as a *model* of ϕ, and we use $\mathsf{models}(\phi)$ to refer to the set of all models of formula ϕ. A formula ϕ is then said to be *satisfiable* if there exists some assignment s that satisfies ϕ, i.e., if $\mathsf{models}(\phi)$ is nonempty, and this is the problem referred to as the Boolean satisfiability (SAT) problem. In other words, Boolean satisfiability is the problem taking as input a propositional formula ϕ and yielding a binary answer stating whether ϕ is satisfiable. Two formulas ϕ and ψ are said to be *equisatisfiable* if ϕ being satisfiable is equivalent to ψ being satisfiable, i.e., either both are are satisfiable or both are unsatisfiable. Note that all equivalent formulas are equisatisfiable, but there are equisatisfiable formulas which are not equivalent. A more thorough treatment of propositional logic can be found in, e.g., [78, 79].

Programs developed to solve the SAT problem are called SAT solvers, and most practical implementations accept their input in a standard form called *conjunctive normal form*. A formula in conjunctive normal (CNF) form uses only three connectives ($\neg$, $\vee$ and $\wedge$) and consists of a conjunction of clauses, i.e., it is of the form

$$\bigwedge_{i \in I} C_i,$$

where C_i is a clause for each $i \in I$. A *clause* then is a disjunction of literals which are atoms p_i or their negations, i.e., a clause is of the form $\bigvee_{i \in I} l_i$ where l_i is either

p_i or $\neg p_i$ for all $i \in I$. CNF is a convenient standard format partly due to the fact that efficient polynomial-time algorithms exist for transforming arbitrary formulas into *linear-size, equisatisfiable* CNF formulas [80, 81].

The significance of the SAT problem stems from the fact that it is an NP-complete problem but admits algorithms that are efficient over inputs encountered in practice. The NP-completeness of SAT refers to it being in the complexity class NP and having the property that every other NP-problem reduces to SAT. Problems in NP are generally characterized as being computationally *intractable* although practically efficient algorithms to NP-complete problems are known. One such example is the CDCL algorithm for SAT [82, 83, 84, 85, 86, 87, 88, 89]. However, before discussing algorithmics we consider the process of modelling problems using propositional logic.

Modelling with Propositional Logic Modelling problems using propositional logic can roughly be split into two parts:

(1) specifying the search space, and

(2) formulating the constraints in CNF.

Part (1) thus consists of deciding the domain of binary variables (i.e. the propositional atoms) over which the constraints are formulated which in practice boils down to naming and deciding the intended meaning of variables the encoding will refer to. Part (2) then consists of producing a propositional formula in CNF that models the constraints required for the problem.

Some constraints are relatively straightforward to encode as clauses. The at-least-one constraint, for example, can be encoded as a single clause $\bigvee_{i \in I} x_i$, which evaluates to true whenever at least one of x_i for $i \in I$ evaluates to true. Other constraints not easily stated in CNF can be formulated using the full set of connectives instead, since the formula can be efficiently translated into an equisatisfiable CNF formula with only linear increase in its size [80, 81].

The following example presents an encoding for finding a 3-coloring of a graph.

Example 3 (Graph coloring). *Let $G = (V, E)$ be a graph. A 3-coloring of G is an assignment of three colors to the nodes of G such that no adjacent nodes have the same color. In other words, G is 3-colorable if there exists a total function $c\colon V \to \{0, 1, 2\}$ such that $c(v) \neq c(u)$ for all $\{v, u\} \in E$.*

To model the situation using propositional logic we take for each $v \in V$ and each $i \in \{0, 1, 2\}$ a variable x_v^i with the intended meaning that v has color i if x_v^i is true. We then construct a Boolean formula $3\text{-col}(G)$ such that any assignment of truth values to variables x_v^i satisfying $3\text{-col}(G)$ corresponds to a 3-coloring of G.

A satisfying assignment to $3\text{-col}(G)$ should

(1) represent a total function $c\colon V \to \{0, 1, 2\}$ assigning one color to each node, and

(2) assign the colors to $v \in V$ such that no adjacent vertices have the same color.

We represent (1) by stating separately that each node $v \in V$ maps to at least one and at most one color $c(v) \in \{0, 1, 2\}$. For each $v \in V$ the encoding includes a clause

$$C_v^{\geq 1} = x_v^0 \lor x_v^1 \lor x_v^2 = \bigvee_{i \in \{0,1,2\}} x_v^i$$

ensuring that at least one of x_v^i for $i \in \{0,1,2\}$ is set true by any satisfying assignment. The encoding contains clauses $\neg x_v^0 \vee \neg x_v^1$, $\neg x_v^0 \vee \neg x_v^2$ and $\neg x_v^1 \vee \neg x_v^2$ for every $v \in V$ to guarantee that every node has at most one color. We denote this as

$$C_v^{\leq 1} = \bigwedge_{\substack{i,j \in \{0,1,2\} \\ i \neq j}} \neg x_v^i \vee \neg x_v^j = (\neg x_v^0 \vee \neg x_v^1) \wedge (\neg x_v^0 \vee \neg x_v^2) \wedge (\neg x_v^1 \vee \neg x_v^2).$$

Clause $\neg x_v^0 \vee \neg x_v^1$, for example, evaluates to false whenever x_v^0 and x_v^1 are set true, thus blocking any satisfying assignment from setting x_v^0 and x_v^1 true simultaneously. Taken together the formulas $C_v^{\geq 1}$ and $C_v^{\leq 1}$ thus ensure that exactly one of $\{x_v^0, x_v^1, x_v^2\}$ is set true by any satisfying assignment. Constraint (1) can therefore be represented as formula

$$F_1 = \bigwedge_{v \in V} \left(C_v^{\geq 1} \wedge C_v^{\leq 1} \right).$$

Constraint (2) can also be encoded using binary clauses since a clause of the form $\neg x_v^i \vee \neg x_u^i$ is satisfied only when either x_v^i or x_u^i evaluates to false. Essentially these clauses block x_v^i and x_u^i from being simultaneously satisfied. The encoding thus contains

$$\neg x_v^i \vee \neg x_u^i$$

for each edge $\{v,u\} \in E$ and each color $i \in \{0,1,2\}$. The formula

$$F_2 = \bigwedge_{\{v,u\} \in E} \bigwedge_{i \in \{0,1,2\}} \left(\neg x_v^i \vee \neg x_u^i \right)$$

thus encodes constraint (2).

The encoding 3-col(G) *is thus the conjunction of formulas F_1 and F_2, i.e.,*

$$\text{3-col}(G) = F_1 \wedge F_2.$$

A problem often encountered in propositional encodings is the existence of symmetries. Generally symmetries refer to transformations of structures that preserve properties of said structures, exemplified by rotations and reflections of geometric figures (square, cube, etc.) which preserve their shape. In the context of Boolean satisfiability the structures are assignments of a formula ϕ and the preserved property is satisfying ϕ. The SAT encoding 3-col(G) from Example 3 contains symmetries as we will show next.

Example 4. *Let $G = (V, E)$ be a graph, and let $c\colon V \to \{0,1,2\}$ be a 3-coloring of G. From c we can easily derive more colorings by changing the names of the colors. Consider for example c' defined as follows.*

$$c'(v) = \begin{cases} 0 & \text{if } c(v) = 1 \\ 1 & \text{if } c(v) = 0 \\ 2 & \text{if } c(v) = 2 \end{cases}$$

Essentially c' is the same as c except that colors 0 and 1 have been swapped. Clearly c' must be a 3-coloring since c is one, but the assignment of 3-col(G) *corresponding to c is*

$$m_c = \bigwedge_{\substack{v \in V \\ c(v)=0}} x_v^0 \wedge \bigwedge_{\substack{v \in V \\ c(v)=1}} x_v^1 \wedge \bigwedge_{\substack{v \in V \\ c(v)=2}} x_v^2$$

whereas the one corresponding to c' is

$$m_{c'} = \bigwedge_{\substack{v \in V \\ c(v)=0}} x_v^1 \wedge \bigwedge_{\substack{v \in V \\ c(v)=1}} x_v^0 \wedge \bigwedge_{\substack{v \in V \\ c(v)=2}} x_v^2.$$

We have now two syntactically different models of 3-col(G) *corresponding to colorings that differ only in naming of the colors. Since c' is a 3-coloring exactly when c is we deduce that $m_{c'}$ must satisfy* 3-col(G) *exactly when m_c does.*

Models m_c and $m_{c'}$ in the previous example are called *symmetric* since they are either both satisfiable or both unsatisfiable. The redundancy of the encoding arising from the existence of symmetric models may be undesirable, but the problem can be mitigated by modifying the encoding. So-called *symmetry-breaking* constraints can be added to the formula resulting in a new formula without symmetric models. We next show one way of breaking symmetry in 3-col(G).

Example 5. *Each 3-coloring of G determines a partition of the set of nodes V into 3 parts such that no adjacent nodes belong to the same part. We break the symmetry by augmenting the formula with constraints that block all but one equivalent 3-colorings. To achieve this we first define an arbitrary linear order $\leq$ over V. The set of colors $\{0, 1, 2\}$ inherits an ordering as a subset of $\mathbb{N}$. We next wish to state that the least node of color 0 precedes the least node of color 1 which precedes the least node of color 2. We use variable y_v^i to mean that node v is the least node of color i. For each color $i \in \{0, 1, 2\}$ the encoding contains clause*

$$\bigvee_{v \in V} y_v^i$$

to ensure that each color has y_v^i set true for at least one node. To guarantee that any node for which y_v^i is set true has the correct color the encoding contains

$$\neg y_v^i \vee x_v^i$$

for each $v \in V$ and $i \in \{0, 1, 2\}$. This binary clause enforces x_v^i to be set true if y_v^i is set true and thus represents the implication $y_v^i \to x_v^i$. The encoding also contains

$$\neg y_{v'}^i \vee \neg x_v^i$$

for each $i \in \{0, 1, 2\}$ and each $v, v' \in V$ such that $v < v'$ which represents the implication $y_{v'}^i \to \neg x_v^i$. These implications essentially state that any preceding node of v' cannot have color i if $y_{v'}^i$ is set true thereby ensuring that the v' for which $y_{v'}^i$ holds is the least node of color i. To enforce the least nodes of each color to be ordered according to the colors the encoding contains clauses

$$\neg y_v^1 \vee \neg y_{v'}^0$$

and

$$\neg y_v^2 \vee \neg y_{v'}^1$$

for each $v, v' \in V$ such that $v < v'$.

The encoding then consists of the formula 3-col(G) *as well as the symmetry-breaking constraints presented here.*

The addition of symmetry-breaking constraints to an encoding as shown in Example 5 is referred to as *static* symmetry breaking [90, 91, 92, 93]. Augmenting an encoding with symmetry-breaking constraints naturally increases the size of the formula, which may result in degraded performance of solving the formula. Compact symmetry-breaking constraints are thus an important research topic in declarative programming.

Including so-called *lex-leader* [90, 91, 92, 93] constraints is one of the simplest ways to break symmetries when encoding problems into CNF. The rough idea is to constrain the model such that only the lexicographically least model out of all equivalent models satisfies the CNF. We use this type of symmetry-breaking constraints in the SAT encoding presented in Section 3.2 to rule out all but one assignment corresponding to each distinct underlying structure, in our case a cycleset of a graph.

Conflict-driven clause learning The most important modern complete algorithm for solving SAT is the *conflict-driven clause learning* algorithm often referred to as CDCL [82, 83, 84, 85, 86, 87, 88, 89]. This algorithm requires the input formula to be in *conjunctive normal form* (CNF).

The main parts of CDCL are *unit propagation*, *conflict learning* and non-chronological backtracking (*backjumping*). Intuitively, CDCL tries to construct an assignment s satisfying the input formula ϕ by assigning a value for each atom in turn and testing whether the assigned values satisfy ϕ. If the values assigned to atoms are sufficient to satisfy ϕ the algorithm terminates with a positive answer. If, on the other hand, the data so far is insufficient to determine the valuation of ϕ under s the algorithm assigns a value for the next atom. Lastly, if the assignment thus-far is enough to determine $s(\phi) = 0$, the algorithm extracts a clause blocking this conflict, adds it to ϕ (**conflict learning**) and **backjumps** far enough to erase the assignment resulting in the conflict. The role of **unit propagation** is to compute the consequences of the current partial assignment in ϕ. For a thorough description of CDCL refer to [21].

The CDCL algorithm implemented using efficient data structures and utilizing appropriate optimizations has proven a very effective tool for declarative programming. Efficient implementations include, e.g., Minisat [94] which we use in this work through the Pysat library [95]. In practice SAT solvers can be used to solve a wide array of computational problems through the *model & solve paradigm*. In this paradigm solving a problem boils down to encoding the desired problem as a propositional formula and using an off-the-shelf SAT solver to find satisfying assignments which correspond to solutions to the problem. All NP-problems can be reduced to SAT due to its NP-completeness, and the user actually implements such a reduction when encoding a problem in propositional logic.

SAT solvers also find uses in solving problems beyond NP via them being used as NP-oracles, i.e., implementing programs that make calls to a SAT solver. An important ingredient of the efficient use of SAT solvers as NP-oracles is *incremental* solving, which refers to saving the state of the solver between repeated calls. This is especially useful when repeated calls are made for an input formula that is only slightly modified between calls, e.g., by adding or deleting clauses.

Model enumeration Model enumeration, sometimes referred to as *all-SAT*, is the problem of finding every satisfying assignment of a propositional formula ϕ [96, 97, 98, 99, 100]. One of the simplest methods for model enumeration is the so-called *blocking* *clause* method which works as follows. To find the models of CNF formula $\phi = \phi_0$ we iterate the process of

(i) finding some model m of ϕ_i, and

(ii) creating a new formula ϕ_{i+1} satisfied exactly by the models of ϕ_i except for m.

Since for each *finite* Boolean formula there are finitely many assignments to consider—that is $2^{\#\text{variables}}$—it follows that the number of satisfying assignments must be $< 2^{\#\text{variables}}$ and thus finite. The above procedure is then guaranteed to terminate since the number of satisfying assignments decreases by one at each step. The formula ϕ_{i+1} is also easily constructed from CNF formula ϕ_i and model m of ϕ_i. Denote by $\mathsf{truelits}(m)$ the collection of literals set true by m. The *so-called* blocking clause $\mathsf{bc}(m)$ of m is the disjunction of negated literals in $\mathsf{truelits}(m)$, i.e.

$$\mathsf{bc}(m) = \bigvee_{l \in \mathsf{truelits}(m)} \neg l.$$

Adding the blocking clause to ϕ_i we then get

$$\phi_{i+1} = \phi_i \wedge \mathsf{bc}(m),$$

which is in CNF since ϕ_i is in CNF and $\mathsf{bc}(m)$ is a clause.

Incremental SAT solving can be easily applied to the blocking clause method since the working formula changes only by the addition of one clause between SAT calls. This increases performance of the method since conflict clauses already learned can be utilized in the subsequent SAT calls without having the relearn them.

While the growth of the working formula can make the blocking clause method unwieldy for certain formulas, it does not present a problem in this work due to the relevant formulas having relatively few models. However, techniques for mitigating the growth of the working formula exist [101, 97] although they are not necessary in this work.

3.2 Enumerating the Cycleset Decompositions of Graphs

In this section we develop a SAT encoding $\phi_{\mathsf{cycles}}(G, g)$ for checking whether a given graph $G = (V, E)$ in $\mathsf{base}(g)$ (for an arbitrary genus $g > 1$) is in $\mathsf{cycles}(g)$. The encoding we formulate also allows for the enumeration of $\mathsf{cyclesets}(G)$ as it captures all cyclesets in $\mathsf{cyclesets}(G)$, i.e., the models of $\phi_{\mathsf{cycles}}(G, g)$ correspond exactly to $\mathsf{cyclesets}(G)$. Existing model enumeration techniques, such as the blocking clause method explained in Section 3.1, can be applied to enumerate $\mathsf{cyclesets}(G)$ for a given graph $G \in \mathsf{base}(g)$.

Recall that a graph G is in $\mathsf{cycles}(g)$ if $G \in \mathsf{base}(g)$ and the directed decomposition of G admits a set of $N = 6(g-1)$ cycles of length 8 (Definition 2). Recall also that $\mathsf{directed}(E)$ denotes the decomposition of undirected edges in E into directed ones, and that the 8-cycles can not have two subsequent directed edges originating from the same undirected edge (see Figure 2.2). Now, finding a cycleset of G boils down to partitioning $\mathsf{directed}(E)$ into N mutually disjoint collections of size 8 while ensuring that each partition forms a cycle. We achieve the partitioning by considering the cycles to be lists of size 8 and using straightforward cardinality constraints. To make sure that the consecutive

$$\text{for each } e \in \mathsf{directed}(E) : \sum_{p \in \{0,\ldots,N-1\}} \mathsf{inCycle}(e,p) = 1 \tag{3.1}$$

$$\sum_{i \in \{0,\ldots,7\}} \mathsf{index}(e,i) = 1 \tag{3.2}$$

$$\text{for each } p \in \{0,\ldots,N-1\} : \bigvee_{e \in \mathsf{directed}(E)} \mathsf{inCycle}(e,p) \tag{3.3}$$

$$\text{for each } p \in \{0,\ldots,N-1\}, i \in \{0,\ldots,7\}, e_1, e_2 \in \mathsf{directed}(E) \text{ such that } e_1 \neq e_2 :$$
$$\neg\mathsf{inCycle}(e_1,p) \vee \neg\mathsf{index}(e_1,i) \vee \neg\mathsf{inCycle}(e_2,p) \vee \neg\mathsf{index}(e_2,i) \tag{3.4}$$

$$\text{for each } v \in V, p \in \{0,\ldots,N-1\}, (e,e') \in \mathsf{pairs}_1(v) :$$
$$\neg\mathsf{orient}(v) \to (\mathsf{inCycle}(e,p) \leftrightarrow \mathsf{inCycle}(e',p)) \tag{3.5}$$

$$\text{for each } v \in V, p \in \{0,\ldots,N-1\}, (e,e') \in \mathsf{pairs}_0(v) :$$
$$\mathsf{orient}(v) \to (\mathsf{inCycle}(e,p) \leftrightarrow \mathsf{inCycle}(e',p)) \tag{3.6}$$

$$\text{for each } v \in V, i \in \{0,\ldots,7\}, i' = (i+1) \bmod 8, (e,e') \in \mathsf{pairs}_1(v) :$$
$$\neg\mathsf{orient}(v) \to (\mathsf{index}(e,i) \leftrightarrow \mathsf{index}(e',i')) \tag{3.7}$$

$$\text{for each } v \in V, i \in \{0,\ldots,7\}, i' = (i+1) \bmod 8, (e,e') \in \mathsf{pairs}_0(v) :$$
$$\mathsf{orient}(v) \to (\mathsf{index}(e,i) \leftrightarrow \mathsf{index}(e',i')) \tag{3.8}$$

$$\text{for each } p \in \{0,\ldots,N-1\}, e, e' \in \mathsf{directed}(E) \text{ such that } e' < e :$$
$$\mathsf{inCycle}(e,p) \wedge \mathsf{index}(e,0) \to \neg\mathsf{inCycle}(e',p) \tag{3.9}$$

$$\text{for each } p \in \{1,\ldots,N-1\}, e, e' \in \mathsf{directed}(E) \text{ such that } e' < e :$$
$$\mathsf{inCycle}(e',p) \wedge \mathsf{index}(e',0) \to \bigwedge_{p'<p} \neg(\mathsf{inCycle}(e,p') \wedge \mathsf{index}(e,0)) \tag{3.10}$$

Figure 3.1: SAT encoding of cyclesets.

edges in each list follow one another we make use of orientations of nodes, and Theorems 5 and 6 stating that each cycleset of G corresponds to an assignment of orientations to the nodes of G. The orientations essentially determine how the incident edges of each node are routed.

We name the N lists using natural numbers $\{0,\ldots,N-1\}$ and consider the lists to have indices $\{0,\ldots,7\}$. We use Boolean variables $\mathsf{inCycle}(e,p)$ to denote that $e \in \mathsf{directed}(E)$ is in cycle (list) $p \in \{0,\ldots,N-1\}$, and the variables $\mathsf{index}(e,i)$ to denote that e is at index $i \in \{0,\ldots,7\}$ of its cycle. We also use Boolean variables $\mathsf{orient}(v)$ to denote which of the two possible orientations node v has.

We list the constraints of the SAT encoding $\phi_{\mathsf{cycles}}(G,g)$ in Figure 3.1. Intuitively the encoding is composed of three parts:

1) ensuring unique assignment of edges into cycles (3.1)–(3.4),

2) enforcing the orientations of nodes to be compatible with the partitioning of $\mathsf{directed}(E)$ (3.5)–(3.8), and

3) breaking certain symmetries present in the encoding otherwise (3.9)–(3.10).

The constraints (3.1)–(3.4) ensure that the elements of $\mathsf{directed}(E)$ are partitioned into the N lists of size 8. Constraints (3.1) intuitively state that each $e \in \mathsf{directed}(E)$ must be assigned to *exactly one* cycle (list) $p \in \{0, \ldots, 7\}$. We encode this constraint using the well-known pairwise encoding which represents the exactly-one constraint using two other constraints: at-least-one and at-most-one. The at-least-one constraint $\sum_p \mathsf{inCycle}(e, p) \geq 1$ corresponds to the clause

$$\bigvee_{p \in \{0, \ldots, N-1\}} \mathsf{inCycle}(e, p)$$

which evaluates to 0 if all of the conjuncts $\mathsf{inCycle}(e, p)$ evaluate to 0, and to 1 otherwise. The at-most-one constraint $\sum_p \mathsf{inCycle}(e, p) \leq 1$ corresponds to a set of binary clauses. For each $p, p' \in \{0, \ldots, N-1\}$ such that $p \neq p'$ the encoding contains clause

$$\neg \mathsf{inCycle}(e, p) \vee \neg \mathsf{inCycle}(e, p')$$

which evaluates to 0 if both $\mathsf{inCycle}(e, p)$ and $\mathsf{inCycle}(e, p')$ evaluate to 1 blocking the simultaneous satisfaction of $\mathsf{inCycle}(e, p)$ and $\mathsf{inCycle}(e, p')$. The pairwise exactly-one encoding results in a quadratic number of binary clauses which does not present issues in this work since the relevant domains considered here are small.

The exactly-one constraints (3.2), stating that each $e \in \mathsf{directed}(E)$ is assigned a unique index, are similarly pairwise encoded. Constraint (3.3), on the other hand, states that the cycles must be non-empty, and is not strictly necessary here since each $e \in \mathsf{directed}(E)$ is constrained to belong to exactly one cycle and the size of $\mathsf{directed}(E)$ is exactly $8N = 48(g-1)$, i.e., the number of cycle-index pairs for the graphs in $\mathsf{base}(g)$, and we do not use the encoding for any other graphs. However, this constraint seems to speed up solving times. Constraint (3.4) blocks two distinct edges $e_1, e_2 \in \mathsf{directed}(E)$ from being assigned to the same partition at the same index: If $e_1, e_2 \in \mathsf{directed}(E)$ are both assigned to cycle p at index i, the clause (3.4) will evaluate to 0. Note that our encoding works with both simple and non-simple graphs as long as edges between the same nodes, i.e., parallel edges, are given distinct names.

Constraints (3.5)–(3.8) encode that the partitions of $\mathsf{directed}(E)$ are cycles, i.e., an edge $e = (x, y)$ at index i in cycle p implies the edge at the next index $(i + 1 \mod 8)$ starts at y. Here we utilize the orientations of nodes to enforce cyclicity. Let us consider the edges incident to a node v, i.e., $\mathsf{edges}(v) = \{e_1, e_2, e_3\}$. We denote the corresponding directed edges by e_i^{out} and e_i^{in} where e_i^{out} refers to the directed edge going outwards from v and vice versa for e_i^{in}. Figure 3.2 illustrates the situation. Additionally, let $O_v(e_1) = e_2$, $O_v(e_2) = e_3$, and $O_v(e_3) = e_1$ correspond to orientation 1, and $O_v(e_1) = e_3$, $O_v(e_2) = e_1$, and $O_v(e_3) = e_2$ correspond to orientation 0 (as in Figure 2.3). We then denote by $\mathsf{pairs}_1(v)$ the successor pairs of the directed edges corresponding to orientation 1, i.e., $\mathsf{pairs}_1(v) = \{(e_1^{\mathrm{in}}, e_2^{\mathrm{out}}), (e_2^{\mathrm{in}}, e_3^{\mathrm{out}}), (e_3^{\mathrm{in}}, e_1^{\mathrm{out}})\}$ and by $\mathsf{pairs}_0(v)$ the successor pairs corresponding to orientation 0, i.e., $\mathsf{pairs}_0(v) = \{(e_1^{\mathrm{in}}, e_3^{\mathrm{out}}), (e_3^{\mathrm{in}}, e_2^{\mathrm{out}}), (e_2^{\mathrm{in}}, e_1^{\mathrm{out}})\}$. These pairs of directed edges are the ones that should be placed at subsequent indices of the corresponding cycles (depending on the assignment of orientations to nodes). The constraint (3.5) then encodes, for each $v \in V$, that the edges appearing in $\mathsf{pairs}_1(v)$ are assigned to the same partition if v has orientation 1 and the constraint (3.6) encodes the same for $\mathsf{pairs}_0(v)$ and orientation 0. Constraint (3.7) ensures that for each $v \in V$ and each $(e, e') \in \mathsf{pairs}_1(v)$ edge e has index i whereas edge e' has the immediately following index $i + 1 \mod 8$ if v has orientation 1, while constraint (3.8) encodes the same for $\mathsf{pairs}_0(v)$ and orientation 0. Essentially these constraints ensure that the incident edges of each node are routed according to the orientation assigned to the node.

The encoding as described so far, i.e., constraints (3.1)-(3.8), is sufficient to ensure that any satisfying assignment indeed corresponds to a cycleset constructed by assigning

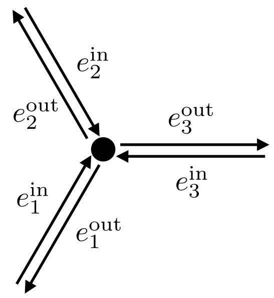


Figure 3.2: Directed decomposition of edges e_1, e_2 and e_3 incident to the same node.

$e \in \mathsf{directed}(E)$ to cycle p at index i if $\mathsf{inCycle}(e, p)$ and $\mathsf{index}(e, i)$ are true under the assignment. However, there are several distinct assignments satisfying constraints (3.1)-(3.8) that correspond to the same cycleset.

Let m be a model of $\phi_{\text{cycles}}(G, g)$ and denote the N 8-cycles corresponding to m by $C_0, \ldots, C_{N-1}$. Each C_i ($i \in \{0, \ldots, N-1\}$) is then a list of size 8 containing directed edges $e \in \mathsf{directed}(E)$. Notice first that we can derive a new model m' of $\phi_{\text{cycles}}(G, g)$ by swapping the names of the cycles as follows. For some chosen pair of cycles C_i, C_j ($i \neq j$) consider the set of cycles where $C'_i = C_j$, $C'_j = C_i$ and $C'_k = C_k$ for all $k \notin \{i, j\}$. The model m' corresponding to this naming of the cycles is constructed from m by swapping the truth values of $\mathsf{inCycle}(e, i)$ and $\mathsf{inCycle}(e, j)$ for all $e \in \mathsf{directed}(E)$. Notice also that lists

$$(e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7)$$

and

$$(e_7, e_0, e_1, e_2, e_3, e_4, e_5, e_6)$$

represent the the same cycle, but with a different indexing of the edges. We can therefore construct models by cyclically permuting the indices of the edges in a cycle, and these models will correspond to the same set of cycles.

The underlying cycleset thus remains unchanged under any permutation of the names of the cycles, i.e., any permutation of $\{0, \ldots, N-1\}$, and similarly the indices of edges in any cycle can be cyclically permuted without changing the cycle. These symmetries do not only degrade performance but make cycleset enumeration very cumbersome due to the huge number of symmetric models. The number of permutations of $\{0, \ldots, N-1\}$ is $N!$ wheres the number of cyclic permutations of the indices is 8^N. The total number of these permutations is thus $8^N N!$ which is of the order 10^8 for genus 2 and 10^{19} for genus 3 (and grows increasingly fast).

Since enumerating all cyclesets of an input graph $G \in \mathsf{base}(g)$ is what we use the encoding for we employ specific symmetry-breaking constraints to block all but one model corresponding to each cycleset. We do this by introducing *ordering constraints* for which we introduce a linear ordering of $\mathsf{directed}(E)$ and use the natural ordering of $\{0, \ldots, N-1\}$. We break the rotational symmetry of the cycle indices by enforcing the edge at index 0 to be the least edge in each cycle. Intuitively constraint (3.9) states that edge e being assigned to cycle p at index 0 blocks the assignment of e' to the same cycle. Since the constraint is stated for each pair of edges (e, e') such that $e' < e$ it effectively enforces the edge at index 0 to be the least edge in cycle p.

To break the permutation symmetry of the cycle names we enforce the cycles $\{0, \ldots, N-1\}$ to be ordered ascendingly according to the edge of index 0 in each cycle, i.e., the edge of index 0 of cycle p is smaller than the edge of index 0 of cycle p' if

$p' > p$. Constraint (3.10) thus states that e' being assigned to cycle p at index 0 blocks the assignment of any larger edge e to index 0 in a preceding cycle $p' < p$.

All in all, the encoding captures $\mathsf{cyclesets}(G)$ for any given G, and in particular, any satisfying assignment of $\phi_{\mathsf{cycles}}(G, g)$ can be projected into a cycleset $W \in \mathsf{cyclesets}(G)$ as follows. Let m be a satisfying assignment to $\phi_{\mathsf{cycles}}(G, g)$. For each $p \in \{0, \dots, N-1\}$ we construct an 8-cycle C_p by finding the $e \in \mathsf{directed}(E)$ such that $m(\mathsf{inCycle}(e, p)) = 1$. These directed edges constitute the 8-cycle C_p. The ordering of these edges can be deduced by considering the head and tail of each directed edge or, alternatively, by finding the indices determined by the encoding. The index of edge e belonging to C_p is the unique i such that $\mathsf{index}(e, i)$ is true in m. We thus define a function $\mathsf{proj}_{\text{cycles}}$ from the models of $\phi_{\mathsf{cycles}}(G, g)$ to $\mathsf{cyclesets}(G)$ such that each $m \in \mathsf{models}(\phi_{\mathsf{cycles}}(G, g))$

$$\mathsf{proj}_{\text{cycles}}(m) = \{C_p \mid p \in \{0, \dots, N-1\}\}$$

where $C_p = (e_0, \dots, e_7)$ and each $e_i \in C_p$ is such that $m(\mathsf{inCycle}(e_i, p)) = 1$ and $m(\mathsf{index}(e_i, i)) = 1$.

Theorem 7 (Correctness of the cycleset encoding)**.** *Let $g > 1$ be a natural number and $G \in \mathsf{base}(g)$. There exists a bijective mapping between the satisfying assignments of $\phi_{\mathsf{cycles}}(G, g)$ and $W \in \mathsf{cyclesets}(G)$.*

Proof. We will show that $\mathsf{proj}_{\text{cycles}}$ as defined previously is the unique bijective mapping from $\mathsf{models}(\phi_{\mathsf{cycles}}(G, g))$ to $\mathsf{cyclesets}(G)$. Let m be an arbitrary satisfying assignment of $\phi_{\mathsf{cycles}}(G, g)$ and $W = \{C_0, \dots, C_{N-1}\}$ its image under $\mathsf{proj}_{\text{cycles}}$. Cardinality constraints (3.1) and (3.2) of $\phi_{\mathsf{cycles}}(G, g)$ ensure that each $e \in \mathsf{directed}(E)$ is associated with exactly one number $p \in \{0, \dots, N-1\}$ (indicating the cycle e belongs to) and one number $i \in \{0, \dots, 7\}$ (indicating the index of e), respectively. Each directed edge $e \in \mathsf{directed}(E)$ thus belongs to a unique cycle C_p at unique index i. The constraint (3.4), on the other hand, ensures that no other e' occupies that same cycle at the same index. Each $C_k \in W$ thus consists of 8 distinct directed edges with no edge belonging to two distinct C_k. The cycles $C_k \in W$ form a partition of $\mathsf{directed}(E)$, since $|\mathsf{directed}(E)| = 48(g-1)$ and W contains $8N = 8 \cdot 6(g-1) = 48(g-1)$ distinct $e \in \mathsf{directed}(E)$.

The model m determines a unique orientation for each $v \in V$ depending on whether $\mathsf{orient}(v)$ is true or false under m. Constraints (3.5)–(3.8) then ensure that each of the directed edges incident to v have a unique predecessor/successor, thus ensuring that $C_k \in W$ are indeed directed cycles.

The symmetry-breaking constraints (3.9) and (3.10) ensure that only one satisfying assignment maps to each cycleset, which makes the projection mapping $\mathsf{proj}_{\text{cycles}}$ injective.

To show that $\mathsf{proj}_{\text{cycles}}$ is surjective, i.e., there exists a model of $\phi_{\mathsf{cycles}}(G, g)$ for each possible $W \in \mathsf{cyclesets}(G)$ it suffices to name the cycles using $\{0, \dots, N-1\}$ and to give consistent indices to the directed edges of each cycle. Constructing a suitable assignment m is then straightforward.

□

Hence, if $\phi_{\mathsf{cycles}}(G, g)$ is unsatisfiable, then $G \notin \mathsf{cycles}(g)$, and if $\phi_{\mathsf{cycles}}(G, g)$ is satisfiable, then $G \in \mathsf{cycles}(g)$ and the satisfying assignments, when projected, yield exactly $\mathsf{cyclesets}(G)$.

3.3 Checking Graphs for a Group Labeling

We next consider whether a graph $G = (V, E) \in \mathsf{walks}(g)$ can be labeled using some $T_i \in \{T_1, \ldots, T_{23}\}$, i.e., whether $G \in \mathsf{labels}(T_i, g)$. We formulate a SAT encoding taking as input a graph G, a group T_i and a cycleset $W \in \mathsf{cyclesets}(G)$ which can be used to decide whether there exists a labeling of G using T_i that is valid with respect to W (see Definitions 3 and 4). We refer to the encoding as $\phi_{\mathsf{labels}}(G, T_i, W)$. If there exists $W \in \mathsf{cyclesets}(G)$ such that $\phi_{\mathsf{labels}}(G, T_i, W)$ is satisfiable, then $G \in \mathsf{labels}(T_i, g)$, whereas if $\phi_{\mathsf{labels}}(G, T_i, W)$ is unsatisfiable for all $W \in \mathsf{cyclesets}(G)$, then $G \notin \mathsf{labels}(T_i, g)$. Recall that $\mathsf{cyclesets}(G)$ can be obtained, e.g., via enumerating the models of the cycleset encoding presented in Section 3.2, or alternatively through orderly generation, as shown later in Chapter 4.

Recall that each of the groups $\{T_1, \ldots, T_{23}\}$ are represented using 15 generators x_i and 15 length–3 relations, which we represent as triplets (x_i, x_j, x_k). A labeling of $G \in \mathsf{cycles}(g)$ using group T_i is a simultaneous labeling of the nodes with triplets and edges with generators of T_i such that each node has a triplet containing the labels of its incident edges. Whether or not a labeling of $G = (V, E)$ using T_i is valid with respect to a cycleset $W \in \mathsf{cyclesets}(G)$ depends on the orientations of $v \in V$ induced by W as well as a chosen 2-coloring of G. Recall that an orientation of $v \in V$ is a cyclic ordering of the edges incident to v. In a valid labeling the triplet of each white v matches the triplet of the labels of its incident edges in the cyclic order defined by W. The same holds for black nodes except that the cyclic ordering is reversed. Additionally the index of the label of edge $e = \{v, u\}$ cannot be the same in the triplets of v and u. In order to simplify the encoding, we reverse the orientations of black nodes arising from the cycleset W and input the adjusted orientations to the encoding as we will explain shortly.

The labeling encoding consists of three parts:

1) ensuring that each edge and node is given a unique label (3.11)–(3.13),
2) enforcing consistency between the labels of each node and its incident edges (3.14), and
3) enforcing consistent labeling of adjacent nodes (3.15)–(3.17).

Before explaining the individual constraints we make the necessary definitions and assumptions.

Assume that $G = (V, E) \in \mathsf{walks}(g)$ for fixed $g > 1$, and let $W \in \mathsf{cyclesets}(G)$ be one of its cyclesets. We denote the generators and relations of a fixed T_i by L and T, respectively. Let (V_B, V_W) be a 2-coloring of G, i.e., $V_B \cup V_W = V$ and $V_B \cap V_W = \emptyset$ such that there is no edge in E consisting of nodes only in V_B or only in V_W. Let $O\colon V \to E \times E \times E$ denote the orientations of nodes as triplets such that if $O(v) = (e_1, e_2, e_3)$, then the orientation of v is the cyclic permutation $(e_1 e_2 e_3)$. Note that triplets (e_2, e_3, e_1) and (e_3, e_1, e_2) refer to the same orientation as (e_1, e_2, e_3). To make the encoding more uniform we define the adjusted orientations $O'\colon V \to E \times E \times E$ by flipping the orientations of black nodes, i.e., $O'(v) = O(v)$ for all $v \in V_W$ and $O'(v) = (O(v)[2], O(v)[1], O(v)[0])$ for all $v \in V_B$. Using the adjusted orientations, items (i) and (ii) in Definition 4 will coincide.

From the adjusted orientation $O'(v) = (e_1, e_2, e_3)$, representing a cyclic ordering of $\mathsf{edges}(v)$, we derive an ordering of the labels of the edges as (l_1, l_2, l_3), where $l_i \in L$ is the label of e_i for $i \in \{1, 2, 3\}$. The label of node v, on the other hand, is also a triplet $t = (x, y, z) \in T$ of elements in L. Since the triplets (y, z, x) and (z, x, y),

$$\text{for each } e \in E: \sum_{l \in L} \mathsf{edgeLabel}(e, l) = 1 \tag{3.11}$$

$$\text{for each } v \in V: \sum_{t \in T} \mathsf{nodeLabel}(v, t) = 1 \tag{3.12}$$

$$\sum_{o \in \{0,1,2\}} \mathsf{offset}(v, o) = 1 \tag{3.13}$$

$$\text{for each } v \in V,\ o \in \{0,1,2\},\ t \in T:$$

$$\mathsf{nodeLabel}(v, t) \wedge \mathsf{offset}(v, o) \rightarrow \bigwedge_{i \in \{0,1,2\}} \mathsf{edgeLabel}(e_i, l_i) \tag{3.14}$$

$$\text{where } e_i = O'(v)[i],\ l_i = t[(i + o) \mod 3]$$

$$\text{for each } \{v_1, v_2\} \in E,\ t = (l_1, l_2, l_3) \in T \text{ such that } l_1 \neq l_2 \neq l_3 \neq l_1:$$

$$\neg\mathsf{nodeLabel}(v_1, t) \vee \neg\mathsf{nodeLabel}(v_2, t) \tag{3.15}$$

$$\text{for each } e = \{v_1, v_2\} \in E,\ t = (l, l, l') \in T \text{ such that } l \neq l':$$

$$\mathsf{nodeLabel}(v_1, t) \wedge \mathsf{nodeLabel}(v_2, t) \rightarrow \mathsf{edgeLabel}(e, l) \tag{3.16}$$

$$\text{for each } e = \{v_1, v_2\} \in E, (o_1, o_2) \in \mathsf{bad\_offsets}^G_W(e), t = (l, l, l') \in T \text{ such that } l \neq l',$$

$$\mathsf{nodeLabel}(v_1, t) \wedge \mathsf{nodeLabel}(v_2, t) \rightarrow \neg\mathsf{offset}(v_1, o_1) \vee \neg\mathsf{offset}(v_2, o_2) \tag{3.17}$$

Figure 3.3: The constraints in the labeling encoding.

represent the same cyclic ordering of the labels, we use a Boolean variable $\mathsf{offset}(v, o)$, where $o \in \{0, 1, 2\}$ to denote which representative of triplet t node v has. Other variables used in the encoding are $\mathsf{edgeLabel}(e, l)$ and $\mathsf{nodeLabel}(v, t)$ denoting that edge $e \in E$ has label (generator) $l \in L$ and node $v \in V$ has label (triplet) $t \in T$.

The constraints of the encoding $\phi_{\mathsf{labels}}(G, T_i, W)$ are shown in Figure 3.3. The cardinality constraints in (3.11)–(3.13) are encoded to clausal form using the pairwise encoding already discussed in Section 3.2. The pairwise encoding is applicable here because the relevant domains, i.e., E, L, V and T are of relatively small size for the graphs and groups we consider. Constraint (3.11) intuitively states that each edge $e \in E$ is assigned exactly one label $l \in L$, whereas constraints (3.12) and (3.13) state that each node $v \in V$ is assigned exactly one triplet $t \in T$ and offset $o \in \{0, 1, 2\}$. The constraint in (3.14) expands to sets of three clauses via straightforward algebraic manipulation of the connectives:

$$\neg\mathsf{nodeLabel}(v, t) \vee \neg\mathsf{offset}(v, o) \vee \mathsf{edgeLabel}(e_0, l_0)$$
$$\neg\mathsf{nodeLabel}(v, t) \vee \neg\mathsf{offset}(v, o) \vee \mathsf{edgeLabel}(e_1, l_1)$$
$$\neg\mathsf{nodeLabel}(v, t) \vee \neg\mathsf{offset}(v, o) \vee \mathsf{edgeLabel}(e_2, l_2)$$

Let $t = (t_0, t_1, t_2)$ be the triplet corresponding to offset 0. Now the triplets corresponding to offsets 1 and 2 are (t_1, t_2, t_0) and (t_2, t_0, t_1), respectively. The edges e_0, e_1, e_2 form the adjusted orientation of node v, i.e., $O'(v) = (e_0, e_1, e_2)$. The labels l_0, l_1 and l_2, on the other hand, form the chosen representative of the triplet of v, i.e., if offset $o = 0$, then $(t_0, t_1, t_2) = (l_0, l_1, l_2)$. If the offset is $o = 1$, then $(t_1, t_2, t_0) = (l_0, l_1, l_2)$, and with offset $o = 2$ we have $(t_2, t_0, t_1) = (l_0, l_1, l_2)$. This is described by the equation

$$l_i = t[(i + o) \mod 3].$$

What the constraints in (3.14) then encode is that if node v has triplet t and offset o,

Table 3.1: Matching triplet t with edges incident to v.

$o_v = 0$	$o_v = 1$	$o_v = 2$
$(t_0, \; t_1, \; t_2)$	$(t_1, \; t_2, \; t_0)$	$(t_2, \; t_0, \; t_1)$
$(e_0, \; e_1, \; e_2)$	$(e_0, \; e_1, \; e_2)$	$(e_0, \; e_1, \; e_2)$

Table 3.2: Matching triplet t with edges incident to u.

$o_u = 0$	$o_u = 1$	$o_u = 2$
$(t_0, \; t_1, \; t_2)$	$(t_1, \; t_2, \; t_0)$	$(t_2, \; t_0, \; t_1)$
$(e_2, \; e_3, \; e_4)$	$(e_2, \; e_3, \; e_4)$	$(e_2, \; e_3, \; e_4)$

then the labels of the edges incident to v (e_0, e_1 and e_2) match the labels of the chosen representative of triplet t.

Constraints in (3.14) together with cardinality constraints in (3.11)–(3.13) rule out cases (d) and (e) in Figure 2.6. In other words these constraints suffice to ensure that edges incident to any node v have labels from the triplet of v and the labels are in the correct order with respect to the given orientation of v.

Constraints in (3.15)–(3.17) together with the cardinality constraints (3.11)–(3.13), on the other hand, rule out cases where adjacent nodes are assigned triples inconsistently, such as case (f) in Figure 2.6. Intuitively, constraint (3.15) blocks any pair $\{v_1, v_2\}$ of adjacent nodes from receiving the same triplet t with three distinct labels. Constraint (3.16) states that connecting edge $e = \{v_1, v_2\}$ of adjacent nodes having the same triplet $t = (l, l, l')$ with only 2 distinct elements has the duplicated label l.

As stated in Definition 4 the position of the label of the connecting edge between two nodes must be different in the triplets of the nodes, see cases (c) and (f) in Figure 2.6.

Example 6. *Let $v, u \in V$ be two adjacent nodes both labeled using triplet $t = (t_0, t_1, t_2)$ with $t_0 = t_1$. Denote the adjusted orientations of v and u by $O'(v) = (e_0, e_1, e_2)$ and $O'(u) = (e_2, e_3, e_4)$, respectively. The offset assigned to v determines the way triplet t and $O'(v)$ are matched together as shown in Table 3.1. Similarly the offset o_u determines the same for node u as shown in Table 3.2.*

If $o_v = 1$ then e_0 has the label t_1, e_1 the label t_2 and e_2 the label t_0. Now e_0 and e_2 have the same label since $t_0 = t_1$, but the label is derived from different position of triplet t.

According to the definition of a valid labeling the connecting edge e_2 must not match the same index in the triples of v and u. In our example this will happen if v has offset $o_v = 1$ and u has offset $o_u = 0$ since e_2 will then match t_0 in the triplet of v as well as the triplet of u. Orientations $o_v = 2$ and $o_u = 1$ result in the same situation except that e_2 matches t_1 in both v and u. Note that the case $o_v = 0$ and $o_u = 2$ leads to e_2 gaining label t_2 from both v and u, but with t_2 not being the repeated label this case is already blocked by constraint (3.16).

The offsets of the nodes along with the chosen triplet determine the positions of the label as shown in Example 6. We thus define the set of illegal pairs of offsets and rule them out as stated in (3.17). We define the set of illegal pairs of offsets for $e \in E$ as

$$\mathsf{bad\_offsets}_W^G(e) = \{(o_1, o_2) \mid e = \{v_1, v_2\}, o_1, o_2 \in \{0,1,2\}, i_1 + o_1 = i_2 + o_2 \mod 3 \\ \text{where } O'(v_1)[i_1] = e \text{ and } O'(v_2)[i_2] = e\}.$$

Here, i_1 and i_2 are the indices of the connecting edge e in the orientations of v_1 and v_2, respectively. The sums $i_1 + o_1$ and $i_2 + o_2 \pmod 3$, on the other hand, represent the indices of the label of e in the triplets of v_1 and v_2, respectively. The set $\mathsf{bad\_offsets}_G(e)$ thus contains the pairs of offsets that would give the label of $e = \{v_1, v_2\}$ the same index in both v_1 and v_2.

Example 7. *Continuing with Example 6 we note that e_2 is the edge connecting $v =: v_1$ and $u =: v_2$. The indices are then $i_1 = 2$ and $i_2 = 0$, since $O'(v) = (e_0, e_1, e_2)$ and $O'(u) = (e_2, e_3, e_4)$. The element of $t = (t_0, t_1, t_2)$ matching e_2 in v is thus $(i_1 + o_v \mod 3) = (2+o_v \mod 3)$ whereas the t_i matching e_2 in u is $(i_2+o_u \mod 3) = o_u \mod 3$. The pairs (o_v, o_u) satisfying equation $2+o_v = o_u \mod 3$ are $(0,2)$, $(1,0)$, and $(2,1)$, which are thus the illegal offsets for v and u, i.e., $\mathsf{bad\_offsets}^G_W(e_2) = \{(0,2), (1,0), (2,1)\}$.*

Next we state the correctness of the labeling encoding $\phi_{\mathsf{labels}}(G, T_i, W)$, and explain briefly why it is correct.

Theorem 8 (Correctness of the labeling encoding)**.** *Let $G \in \mathsf{cycles}(g)$ for $g > 1$, and T_i be one of the 23 groups constructed in [57] . Then $G \in \mathsf{labels}(T_i, g)$ iff there exists $W \in \mathsf{cyclesets}(G)$ such that $\phi_{\mathsf{labels}}(G, T_i, W)$ is satisfiable.*

Proof sketch. Cardinality constraints (3.11)-(3.13) ensure that each node and edge is assigned a unique label. The cardinality constraints together with (3.14) blocks any assignment of labels to a node and its incident edges that is inconsistent with the orientations arising from cycleset W. Lastly the cardinality constraints along with (3.15)-(3.17) block any illegal labeling of adjacent nodes. □

Observe that a valid labeling for G with respect to W using T_i can be directly extracted from a satisfying assignment to $\phi_{\mathsf{labels}}(G, T_i, W)$, which allows one to construct the periodic apartment in the hyperbolic building corresponding to T_i that is invariant under the action of a genus g surface. Projecting a satisfying assignment of $\phi_{\mathsf{labels}}(G, T_i, W)$ into a labeling of $G = (V, E)$ is straightforward.

Assume that m is a model of $\phi_{\mathsf{labels}}(G, T_i, W)$. Now the model m sets exactly one variable $\mathsf{nodeLabel}(v, t)$ true for each node $v \in V$, due to cardinality constraints (3.12) in Figure 3.3, and the $t \in T$ for which $m(\mathsf{nodeLabel}(v, t)) = \mathsf{true}$ becomes the label of v. Similarly, due to cardinality constraints (3.11), m sets exactly one variable $\mathsf{edgeLabel}(e, l)$ true for each edge $e \in E$, and again the label l that makes $\mathsf{edgeLabel}(e, l)$ true in m becomes the label of e.

Chapter 4

Orderly Generation of Graphs and Their Cyclesets

In this chapter we develop an orderly algorithm for directly generating the graphs admitting cycleset decompositions. This algorithm also enumerates the distinct cycleset decompositions of each graph it generates. Before developing our specialized algorithm we give an overview of an off-the-shelf orderly generation program we utilize in this thesis.

Multigraph is a program that generates exhaustive lists of connected (multi)graphs of a given size and degree sequence, and it has an option to generate only bipartite graphs [76]. The program is thus suitable for generating $\mathsf{base}(g)$ (for $g > 1$), i.e., bipartite, 3-regular, connected graphs on $16(g-1)$ nodes and $24(g-1)$ edges. There are no publications about Multigraph, but another graph generator called Minibaum [71, 102] utilizes similar methods. The principal difference between Minibaum and Multigraph is the fact that Minibaum can generate only simple graphs whereas Multigraph is able to generate non-simple graphs as well. Both generators are based on orderly generation [22].

However, a large majority of graphs in $\mathsf{base}(g)$ does not belong to $\mathsf{cycles}(g)$. We therefore develop an algorithm for the exhaustive generation of graphs in $\mathsf{cycles}(g)$ and $\mathsf{cyclesets}(G)$ for each $G \in \mathsf{cycles}(g)$. We achieve this by generating *configurations* which can be mapped to pairs (G, W) where $G \in \mathsf{cycles}(g)$ and $W \in \mathsf{cyclesets}(G)$. The algorithm is based on Read's *orderly generation* method [22] and we develop optimizations specific to our configurations. In orderly generation the explicit removal of duplicates in the isomorph-free collection is avoided by generating configurations in a specific *order* and outputting only canonical configurations greater than the previous one.

4.1 Towards an Orderly Algorithm

Central notions required to develop an orderly generation algorithm are a formal definition of a *configuration* and their *linear ordering*, a notion of *canonicity* as well as a depth parameter and a method of augmentation. The idea of the construction is to take $N = 6(g-1)$ bipartite directed cycles of length 8 and glue the directed edges together (into undirected edges) while preserving the connectedness and bipartiteness of the induced graph and ensuring that the resulting graph is 3-regular. We number the cycles with elements $c \in C = \{0, \ldots, N-1\}$, and in each cycle, its directed edges with $i \in I = \{0, \ldots, 7\}$. Hence, each directed edge is identified by a pair $(c, i) \in C \times I$. We set the source node of each directed edge at even (odd) index black (white) to indicate bipartiteness. In other words the source nodes of directed edges $(c, 0)$, $(c, 2)$, $(c, 4)$ and

$(c, 6)$ are black and the source nodes of directed edges $(c, 1)$, $(c, 3)$, $(c, 5)$ and $(c, 7)$ are white.

Formally a configuration $\mathcal{X}$ is a list of ordered pairs $x_i = \langle e_1, e_2 \rangle$ of edge identifiers $e_1 = (c_1, i_1)$ and $e_2 = (c_2, i_2)$, representing the directed edges that have been glued together to form undirected edges. We take the depth parameter to be the length of a configuration. There are $8N$ distinct edge identifiers since there are N 8-cycles, and therefore the maximal length of a configuration is $4N$. Denote by $\mathcal{C}(N)$ the set of configurations of length at most $4N$ built from N 8-cycles. We define an ordering of $\mathcal{C}(N)$ by lifting the natural lexicographic ordering of edge identifiers $e = (c, i)$ to lists of pairs of edge identifiers, i.e., configurations. Thus $e_1 = (c_1, i_1) \leq e_2 = (c_2, i_2)$ if $c_1 < c_2$ or $c_1 = c_2$ and $i_1 < i_2$. Now $x = \langle e_1, e_2 \rangle \leq x' = \langle e_1', e_2' \rangle$ if $e_1 < e_1'$ or $e_1 = e_1'$ and $e_2 < e_2'$. The ordering of configurations takes into account their varying length, i.e., $\mathcal{X} = (x_0, \ldots, x_l) \leq \mathcal{Y} = (y_0, \ldots, y_k)$ if (i) $\mathcal{X} = \mathcal{Y}$, (ii) $l < k$, or (iii) $l = k$ and $\exists i$ such that $x_i < y_i$ and $x_j = y_j$ for all $j < i$.

Observe that configurations $\mathcal{X}, \mathcal{Y} \in \mathcal{C}(N)$ may be syntactically different representations of the same graph-cycleset pair. Specifically, the names of cycles and their edges bear no relevance to the induced graph-cycleset pair. We may thus permute the cycle names arbitrarily and the indices of edges in steps of two (due to bipartiteness). Stated group-theoretically, the symmetry group of $\mathcal{C}(N)$ is the direct product of the symmetric group on N elements S_N (corresponding to permuting the names of cycles) and the N-wise product of the cyclic group of 4 elements (corresponding to permuting the indices). Each group element $\pi = (\pi', o_0, \ldots, o_{N-1}) \in S_N \times C_4^N$ thus consists of a permutation of cycle names $\pi' \in S_N$ as well a cyclic permutation $o_c \in C_4$ of indices $\{0, \ldots, 7\}$ (in steps of two) of each cycle $c \in \{0, \ldots, N-1\}$. In the following definition we state how elements of $S_N \times C_4^N$ act on individual edges $e = (c, i)$.

Definition 6 (Effect of $S_N \times C_4^N$ on edge identifiers)**.** *Let $\pi = (\pi', o_0, \ldots, o_{N-1}) \in S_N \times C_4^N$ and (c, i) be an edge identifier, i.e., $c \in \{0, \ldots, N-1\}$ and $i \in \{0, \ldots, 7\}$. The effect of group element $\pi = (\pi', o_0, \ldots, o_{N-1})$ on edge identifier (c, i) is defined as*

$$\pi(c, i) = (\pi'(c), o_c(i)).$$

Definition 6 states that the permutation $\pi' \in S_N$ of $\{0, \ldots, N-1\}$ affects the cycle name whereas the index is mapped by the cyclic permutation o_c chosen from $o_0, \ldots, o_{N-1}$ according to which cycle edge (c, i) belongs to. Now, supplied with the action of $S_N \times C_4^N$ on individual edge identifiers, we are ready to lift the action of $S_N \times C_4^N$ to the level of pairs of edges and further to the level of configurations.

Definition 7 (Action of $S_N \times C_4^N$ on configurations)**.** *Let $\pi = (\pi', o_0, \ldots, o_{N-1}) \in S_N \times C_4^N$ and $\mathcal{X} = (x_0, \ldots x_l)$ where $x_i = \langle e_i^1, e_i^2 \rangle$ for all $i \in \{0, \ldots, l\}$. Now the effect of $\pi = (\pi', o_0, \ldots, o_{N-1})$ on configuration $\mathcal{X}$ is defined as*

$$\pi(\mathcal{X}) = \mathsf{sort}(\pi(x_0), \ldots \pi(x_l)),$$

where $\pi(x_i) = \mathsf{sort}(\langle \pi(e_i^1), \pi(e_i^2) \rangle)$ for all $i \in \{0, \ldots, l\}$.

Intuitively, Definition 7 states that a permutation $\pi \in S_N \times C_4^N$ acts on the pairs of edges by mapping the individual edges while ensuring lexicographic ordering. The configuration itself is then mapped by applying π to the individual pairs of $\mathcal{X}$ which are then arranged according to lexicographic order. Sorting the pairs and the list is vital

Table 4.1: Permuting a configuration step by step in Example 8.

pair	**edges permuted**	**pair sorted**
$\langle(0,0),(1,1)\rangle$	$\langle(1,6),(0,1)\rangle$	$\langle(0,1),(1,6)\rangle$
$\langle(0,1),(1,2)\rangle$	$\langle(1,7),(0,2)\rangle$	$\langle(0,2),(1,7)\rangle$
$\langle(0,2),(0,7)\rangle$	$\langle(1,0),(1,5)\rangle$	$\langle(1,0),(1,5)\rangle$
$\langle(1,0),(1,3)\rangle$	$\langle(0,0),(0,3)\rangle$	$\langle(0,0),(0,3)\rangle$

for ensuring that the permuted configurations are, indeed, valid configurations according to our definitions. Keeping the configurations lexicographically ordered helps us avoid certain symmetries in the linear representation of configurations.

Now two configurations $\mathcal{X}$ and $\mathcal{Y}$ are equivalent if there exists a permutation $\pi \in S_N \times C_4^N$ such that $\mathcal{X} = \pi(\mathcal{Y})$. Stated in group-theoretic terms, the equivalence classes of configurations are exactly the orbits of $S_N \times C_4^N$ acting, as stated in Definition 7, on the set of configurations. We denote by $[\mathcal{X}]$ the equivalence class of $\mathcal{X} \in \mathcal{C}(N)$, i.e., $[\mathcal{X}] = \{\mathcal{Y} \in \mathcal{C}(N) \mid \mathcal{Y} \text{ and } \mathcal{X} \text{ are equivalent}\}$. To obtain a unique, canonical, representative of each equivalence class, we take the least configuration according to the lexicographic order. In other words, the canonical representative of $[\mathcal{X}]$ is the least $\mathcal{Y} \in [\mathcal{X}]$.

Example 8 (Equivalent configurations)**.** *Let us map configuration*

$$\mathcal{X} = (\langle(0,0),(1,1)\rangle, \langle(0,1),(1,2)\rangle, \langle(0,2),(0,7)\rangle, \langle(1,0),(1,3)\rangle)$$

*using permutation $\pi \in S_N \times C_4^N$ swapping cycles 0 and 1 with cyclic offsets $o_0 = -2$ and $o_1 = 0$. Permutation π thus maps edges $(0,i)$ to $(1, i-2 \mod 8)$ and edges $(1,j)$ to $(0,j)$ for all $i,j \in \{0,\dots,7\}$. When applying π to a pair $\langle e,e'\rangle$ the edges are first permuted after which the pair must be sorted to make sure it is in lexicographic order. Table 4.1 lists the results of applying π to each pair in $\mathcal{X}$. The middle column (**edges permuted**) shows the pair after applying π to the edges but before sorting has been done.*

*Lastly we collect the permuted pairs, i.e., the entries in the rightmost column (**pair sorted**), and order them lexicographically to get*

$$\mathcal{Y} = \pi(\mathcal{X}) = (\langle(0,0),(0,3)\rangle, \langle(0,1),(1,6)\rangle, \langle(0,2),(1,7)\rangle, \langle(1,0),(1,5)\rangle)$$

Now $\mathcal{X}$ and $\mathcal{Y}$ are equivalent configurations since $\mathcal{Y}$ is the result of applying π to $\mathcal{X}$. Observe also that $\mathcal{Y} < \mathcal{X}$ because $\langle(0,0),(0,3)\rangle < \langle(0,0),(1,1)\rangle$.

4.2 Projecting Configurations into Graph-cycleset Pairs

Before delving into the details of the orderly generation algorithm we first consider how the underlying graph and orientations of its nodes can be extracted from a configuration. Recall that each cycleset corresponds to an assignment of orientations to nodes of the graph as shown in Theorem 6. We describe how to extract the orientations directly since they are required by the labeling encoding described in Section 3.3.

Assume that $\mathcal{X} = (x_0, \dots, x_{l-1}) \in C(N)$ is a configuration of length $l = 4N$ constructed from $N = 6(g-1)$ (for a fixed $g > 1$) directed bipartite 8-cycles. Each edge identifier $e = (c,i)$ refers to a directed edge, and thus each pair $x_i = \langle e,e'\rangle$ corresponds

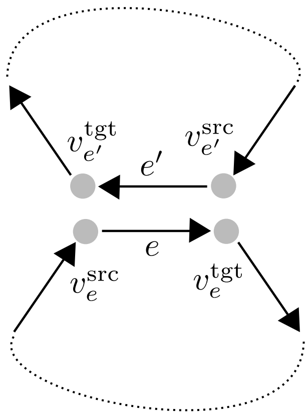


Figure 4.1: Pair $\langle e, e' \rangle$ in a configuration implies that v_e^{src} is identified with $v_{e'}^{\mathrm{tgt}}$ and $v_{e'}^{\mathrm{src}}$ with v_e^{tgt}.

to an undirected edge formed by attaching directed edges e and e' pointing in opposite directions. We denote the source and target nodes of e by v_e^{src} and v_e^{tgt}, respectively. The source node v_e^{src} of every $e = (c, i)$ with even index i is colored black whereas the source node of every odd-indexed edge is colored white. Now attaching edges $e = (c, i)$ and $e' = (c', i')$ also implies that the source node of e is identified with the target node of e' and vice versa, i.e., node v_e^{src} is identified with $v_{e'}^{\mathrm{tgt}}$ and $v_{e'}^{\mathrm{src}}$ is identified with v_e^{tgt}, see Figure 4.1.

We denote by V_{conf} the nodes of the 8-cycles, i.e., $V_{\mathrm{conf}} = \{v_e^{\mathrm{src}} \mid e \in C \times I\}$. Using the procedure just described we partition V_{conf} by iterating through elements $x_0, \ldots, x_{l-1}$ and identifying the nodes as described. We enforce transitivity to actually obtain a partition, i.e., having identified v with u and u with w we also identify v with w. We denote the partitions $\mathsf{part}(V_{\mathrm{conf}}) = \{[v] \mid v \in V_{\mathrm{conf}}\}$ with $[v]$ being the unique partition containing v. Each equivalence class $[v]$ now corresponds to a node of the underlying graph with the size of $[v]$ being equal to its degree (since each identified node contributes 2 halves of an undirected edge).

The graph $G = (V, E)$ underlying configuration $\mathcal{X}$ is constructed as follows. We set $V = \mathsf{part}(V_{\mathrm{conf}})$ making each $v \in V$ a set of identified nodes of the 8-cycles. Each pair $x_i = \langle e, e' \rangle$ of $\mathcal{X}$ corresponds to single undirected edge, and the incident nodes of this undirected edge are found as the equivalence classes of the source nodes of e and e' as illustrated in Figure 4.2. To construct the multiset of edges E we therefore iterate through $\mathcal{X}$, and for each pair $\langle e, e' \rangle$ add edge $\{[v_e^{\mathrm{src}}], [v_{e'}^{\mathrm{src}}]\}$ to E.

What remains to be done is to determine the orientation of each $v \in V$. Denote the incident undirected edges of $v \in V$ by e_i^* for $i \in \{1, 2, 3\}$. Denote each incoming half of

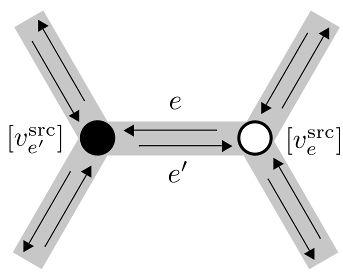


Figure 4.2: Finding the orientation of a node.

Algorithm 1: Orderly generation algorithm $\mathsf{orderly}(\mathcal{X}, l, N)$

Input: configuration $\mathcal{X}$, level l, number of 8-cycles N
Output: the list of maximal length canonical representatives of $\mathcal{C}(N)$ having $\mathcal{X}$ as their prefix

```
G ← ()                                                  /* empty list */
if l = 4N then
    if canonical(X) then G ← append(G, X)
else
    A ← augmentations(X)
    while A ≠ () do
        a ← pop(A)   /* returns first element while removing from A */
        X′ ← append(X, a)
        if canonical(X′) then G ← append(G, orderly(X′, l + 1, N))
return G
```

e_i^* by e_i and each outgoing half by $\bar{e}_i$ as in Figure 2.13. We deduce that either e_1 and $\bar{e}_2$ or e_1 and $\bar{e}_3$ must originate from the same cycle since each directed edge is belongs to an 8-cycle. The above choice in fact determines the orientation of v. If e_1 and $\bar{e}_2$ originate from the same cycle, then so must pairs $(e_2, \bar{e}_3)$ and $(e_3, \bar{e}_1)$ in which case the orientation of v given as a cyclic permutation is (e_1^*, e_2^*, e_3^*). In the latter case it is deduced similarly that v must have orientation (e_1^*, e_3^*, e_2^*).

4.3 Structuring the Orderly Algorithm

Our orderly generation algorithm is outlined as Algorithm 1. The recursive algorithm starts with an initial configuration $\mathcal{X}$ of length l and incrementally constructs a list of canonical configurations of maximal length $(4N)$ having $\mathcal{X}$ as their prefix. Hence, starting from the empty configuration, we obtain the list of canonical configurations built from N 8-cycles. The main parts of the algorithm are the *augmenting* and *canonicity checking* steps. Augmenting adds a new pair $\langle e_1, e_2 \rangle$ to the end of configuration $\mathcal{X}$ thereby increasing its length by 1. Canonicity checking consists of determining whether a given configuration is the canonical representative of its equivalence class, and is performed greedily after each augmentation step since this efficiently prunes the search space.

For the algorithm to work correctly, it needs to output exactly one configuration from each equivalence class of configurations, and this representative configuration should be the canonical one. It is straight-forward to see that our algorithm satisfies the following conditions, which are a slightly stronger variant of Read's necessary and sufficient conditions [22] for the correctness of orderly generation.

(i) Each canonical configuration of length $q + 1$ can be produced from exactly one canonical configuration of level q.

(ii) If $\mathcal{X}$ and $\mathcal{Y}$ are configurations of length q and $\mathcal{X} < \mathcal{Y}$, the canonical configurations produced by augmenting $\mathcal{X}$ must precede the ones produced by augmenting $\mathcal{Y}$.

(iii) The augmenting operation produces the new configurations in order.

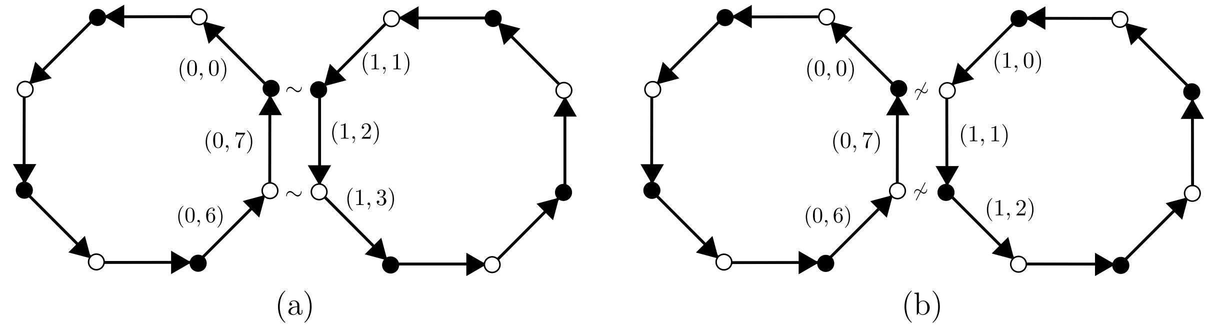


Figure 4.3: (a) Glueing edges $(0, 7)$ and $(1, 2)$ is allowed, since one index is odd while the other is even. (b) Glueing $(0, 7)$ to $(1, 1)$ is not allowed, since both indices are odd.

4.4 Augmenting Configurations

Let $\mathcal{X} = (x_0, \dots x_{l-1})$ be a configuration of length l. The augmenting subroutine produces a list of pairs $\langle e_1, e_2 \rangle$ used in the main algorithm to extend $\mathcal{X}$, denoted by $\mathsf{augmentations}(\mathcal{X})$. To ensure correctness the list $\mathsf{augmentations}(\mathcal{X})$ should contain every element $p = \langle e_1, e_2 \rangle$ for which $\mathsf{append}(\mathcal{X}, p)$ is canonical. Note that if $\mathsf{augmentations}(\mathcal{X})$ contains elements yielding non-canonical configurations, correctness is still maintained, but empirical performance may degrade. Also note that any canonical configuration not having $\mathcal{X}$ as a prefix will be produced by augmenting some other suitable configuration $\mathcal{X}'$.

We denote the free edges of $\mathcal{X}$ by $E^{\mathsf{free}}_{\mathcal{X}} = (C \times I) \setminus \mathsf{edges}(\mathcal{X})$, where $\mathsf{edges}(\mathcal{X})$ consists of all the edge identifiers appearing in the ordered pairs $x_i \in \mathcal{X}$. The list $\mathsf{augmentations}(\mathcal{X})$ thus consists of pairs $\langle e_1, e_2 \rangle$ where $e_1, e_2 \in E^{\mathsf{free}}_{\mathcal{X}}$. For each pair, we set e_1 to the least element in $E^{\mathsf{free}}_{\mathcal{X}}$, denoted by e_{min}. For e_2 we consider a subset E' of the remaining elements $E^{\mathsf{free}}_{\mathcal{X}} \setminus \{e_{\mathsf{min}}\}$, i.e., $\mathsf{augmentations}(\mathcal{X}) = (\langle e_1, e_2 \rangle \mid e_1 = e_{\mathsf{min}}, e_2 \in E')$ with the pairs listed in order.

For $E' \subseteq E^{\mathsf{free}}_{\mathcal{X}} \setminus \{e_{\mathsf{min}}\}$ we observe the following. Denote $e_1 = (c_1, i_1)$, $e_2 = (c_2, i_2)$, and let $c_{\max}$ be the largest cycle number appearing in $\mathcal{X}$. To ensure bipartiteness we include in E' only edges e_2 such that i_2 has different parity to i_1, see Figure 4.3 for examples. If $\mathsf{edges}(\mathcal{X}) = \{0, \dots, c_{\max}\} \times I$, then all the edges currently in $\mathcal{X}$ have already been paired. In this case augmenting with a new pair would yield a configuration with a disconnected underlying graph, and hence if $\mathsf{edges}(\mathcal{X}) = \{0, \dots, c_{\max}\} \times I$, we set $E' = \emptyset$. Finally, E' is reduced by observing that we need to ensure that any partial configuration can be augmented to a full configuration so that the underlying graph is 3-regular. Two types of nodes can prevent this: (i) nodes with degree > 3 or (ii) nodes with degree 1 or 2 such that their degree cannot be increased by augmenting. Hence we exclude from E' all edges that would result in such nodes in the underlying graph, guaranteeing the 3-regularity of the induced graph in a configuration of maximal length $4N$.

We ensure eventual 3-regularity by determining which pairs $\langle e_1, e_2 \rangle$ when added to $\mathcal{X}$ would create a node of type (i) or (ii) as follows. Let V be the set of nodes of the directed 8-cycles, i.e., $V = \{v^{\mathrm{src}}_e | e \in C \times I\}$. Let $v \in V$ be the node between two subsequent edges e, e', i.e., $e = (c, i)$, $e' = (c, i^{\mathrm{next}})$ and $v = v^{\mathrm{tgt}}_e = v^{\mathrm{src}}_{e'}$. Node v is called *blocked* in $\mathcal{X}$ if both e and e' are present in $\mathcal{X}$, i.e., $e, e' \in \mathsf{edges}(\mathcal{X})$, see Figure 4.4. Recall that the underlying graph of $\mathcal{X}$ is constructed by considering which nodes $v \in V$ are identified in the process of attaching pairs of undirected edges to create directed edges. The nodes $v \in V$ thus form equivalence classes with each class corresponding to a single of the underlying graph.

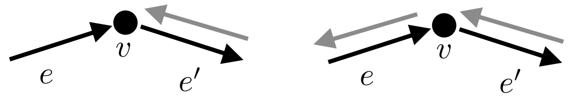


Figure 4.4: On the right node v is blocked since both e and e' are paired with some directed edges. On the left node v is not blocked.

The size of each equivalence class $[v] \subset V$ then corresponds to the degree of node. We call an equivalence class $[v]$ blocked in $\mathcal{X}$ if every node $v' \in [v]$ is blocked in $\mathcal{X}$. A blocked $[v]$ corresponds to a node of the underlying graph whose degree cannot be increased by augmenting, whereas non-blocked $[v]$ correspond to nodes whose degree may increase by further augmentations.

For each $e_2 \in E'$ we construct the equivalence classes of nodes in V for $\mathcal{X}' = \mathsf{append}(\mathcal{X}, \langle e_1, e_2 \rangle)$ and check which equivalence classes $[v] \subset V$ are blocked in $\mathcal{X}'$. We then iterate through the equivalence classes $[v]$ checking two properties corresponding to (i) and (ii). If an equivalence class $[v]$ of size > 3 is found we remove e_2 from E' to prevent nodes of type (i). On the other hand, if a blocked equivalence class $[v]$ of size 1 or 2 exists, we reject e_2 to prevent nodes of type (ii).

Observe that while these considerations are enough to guarantee that only augmentations yielding connected, bipartite, eventually 3-regular underlying graphs are produced, E' can be reduced even further by excluding pairs that would necessarily yield a non-canonical configuration.

Let c_{max} be the largest cycle number appearing in $\mathcal{X}$. Any augmentation of $\mathcal{X}$ with $\langle (c_1, i_1), (c_2, i_2) \rangle$, where $c_2 > c_{\mathsf{max}}+1$ cannot be canonical, since a lexicographically smaller one is obtained by swapping c_2 and $c_{\mathsf{max}} + 1$ (and such a symmetry exists in $S_N \times C_4^N$). Hence, we may remove any edges belonging to cycles $c > c_{\mathsf{max}} + 1$ from E' thereby restricting E' to be a subset of $\{0, \ldots, c_{\mathsf{max}} + 1\} \times I$. Also, since any edge identifier $(c_{\mathsf{max}} + 1, i)$, where $i > 1$ may be mapped to either $(c_{\mathsf{max}} + 1, 0)$ or $(c_{\mathsf{max}} + 1, 1)$ using a permutation that does not move edges in any other cycle, it suffices to consider only indices 0 and 1 with $c_{\mathsf{max}} + 1$.

Furthermore, edges in the same cycle are allowed to be glued together, and this will produce a configuration corresponding to a graph with one or more double edges. Since each cycle may have at most one self-attachment due to 3-regularity, if $e_1 = (c, i)$ is the first element of the pairs in $\mathsf{augmentations}(\mathcal{X})$ and cycle c already has a self-attachment, we may remove all edges (c, j) from E'.

Example 9. *Let* $\mathcal{X} = (\langle (0,0), (0,3) \rangle, \langle (0,1), (1,0) \rangle)$. *Now*

$$\mathsf{edges}(\mathcal{X}) = \{(0,0), (0,1), (0,3), (1,0)\}$$

and $c_{max} = 1$. *The free edges of* $\mathcal{X}$ *are thus*

$$E_{\mathcal{X}}^{free} = (\{0,1,2\} \times \{0, \ldots, 7\}) \setminus \{(0,0), (0,1), (0,3), (1,0)\}$$

with the least free edge being $\min E_{\mathcal{X}}^{free} = (0,2)$. *We therefore set* $e_1 = (0,2)$ *and for the second edge* e_2 *we consider all* $(c,i) \in E_{\mathcal{X}}^{free}$ *for which* i *is odd (to ensure bipartiteness). Since cycle* 0 *already has a self-attachment in* $\mathcal{X}$, *i.e., pair* $\langle (0,0), (0,3) \rangle$, *we need not consider any edges from cycle* 0, *and due to cycle* 2 *not being present in* $\mathcal{X}$ *it suffices to consider only edge* $(2,1)$ *of cycle* 2. *So far we have trimmed the set* $E' \subset E_{\mathcal{X}}^{free} \setminus \{e_1\}$ *down to* $\{(1,1), (1,3), (1,5), (1,7), (2,1)\}$, *but we must trim it even further to guarantee eventual* 3*-regularity.*

Algorithm 2: Canonicity checking algorithm, $\mathsf{canonical}(\mathcal{X})$

Input: configuration $\mathcal{X}$
Output: Boolean indicating whether $\mathcal{X}$ is canonical
$c_{\mathsf{max}} \leftarrow$ largest cycle number in $\mathcal{X}$
for $c \in \{0, \ldots, c_{\mathsf{max}}\}$ **do**
  for $a \in \{0, 2, 4, 6\}$ **do**
    $\pi \leftarrow \{\langle (c, i), (0, i + a \bmod 8)\rangle \mid i \in \{0, \ldots, 7\}\}$
    $\pi \leftarrow \mathsf{extend}(\pi, \mathcal{X})$ `/* extend` π `to a full permutation */`
    $\mathcal{Y} \leftarrow \pi(\mathcal{X})$ `/* compute` $\mathcal{Y}$ `*/`
    if $\mathcal{Y} < \mathcal{X}$ **then return** *false*
return *true* `/* No equivalent configuration smaller than` $\mathcal{X}$ `was found */`

Denote the source node of edge (c, i) *by* v_i^c*. Due to pair* $\langle (0, 0), (0, 3)\rangle$ *we must identify* v_0^0 *with* v_4^0 *and* v_1^0 *with* v_3^0 *and due to* $\langle (0, 1), (1, 0)\rangle$ *we must identify* v_1^0 *with* v_1^1 *and* v_2^0 *with* v_0^1*. The equivalence classes of nodes of* $\mathcal{X}$ *are thus* $\{v_0^0, v_0^4,\}$*,* $\{v_2^0, v_0^1\}$ *and* $\{v_1^0, v_3^0, v_1^1\}$ *with the rest being singletons. The only blocked node of* $\mathcal{X}$ *is* v_1^0 *since both* $(0, 0)$ *and* $(0, 1)$ *are in* $\mathcal{X}$*. None of the equivalence classes of* $\mathcal{X}$ *are therefore blocked. Consider adding* $\langle (0, 2), (2, 1)\rangle$ *to* $\mathcal{X}$*. Due to the new pair we need to identify* v_2^0 *with* v_2^2 *and* v_3^0 *with* v_1^2 *yielding the equivalence classes (omitting singletons)* $\{v_0^0, v_0^4,\}$*,* $\{v_2^0, v_0^1, v_2^2\}$ *and* $\{v_1^0, v_3^0, v_1^1, v_1^2\}$*. Since there is now an equivalence class of size* 4 *we must reject the choice* $e_2 = (2, 1)$*.*

Using similar reasoning we must in fact reject $(1, 7)$*,* $(1, 5)$ *and* $(1, 3)$ *leaving only* $(1, 1)$*. Hence* $\mathsf{augmentations}(\mathcal{X}) = \{\langle (0, 2), (1, 1)\rangle\}$*.*

4.5 Checking Canonicity of Configurations

It is not known whether graph isomorphism in general is polynomial-time computable, but in the restricted case of degree-bounded graphs the isomorphism problem is, in fact, polynomial-time computable [103]. In our case, the graphs underlying configurations are 3-regular, and hence it is not surprising that configurations can be canonized in polynomial-time.

We outline our canonicity checking procedure in Algorithm 2. Given a configuration $\mathcal{X} = (x_0, \ldots x_{q-1})$ of length q the algorithm iterates through $\mathcal{X}$ while constructing the permutation minimizing the configuration. If this permutation produces a configuration smaller than $\mathcal{X}$, we conclude that $\mathcal{X}$ is not canonical. The algorithm goes through permutations mapping each cycle to 0 with different offsets. Once it is fixed which cycle maps to 0 and with which offset, there is only one way to extend the permutation in a way that minimizes $\mathcal{Y} = \pi(\mathcal{X})$, and hence it suffices to iterate through every value for c and a instead of trying every element of $S_N \times C_4^N$. Here, an essential observations is that any configuration $\mathcal{X}$ produced by the augmentation in our algorithm is connected.

The extending of a partial permutation π in $\mathsf{extend}(\pi, \mathcal{X})$ works as follows. Assume that permutation π maps cycle c to 0 with offset a. The pairs of $\mathcal{X}$ containing edges from cycle c will become the prefix of the permuted configuration $\mathcal{Y} = \pi(\mathcal{X})$ since 0 is the least cycle. Let $\mathsf{cindex}(\pi, \mathcal{X}) = \{c\}$ and $\mathsf{cindex}(\pi, \mathcal{Y}) = \{0\}$ be the sets of cycle indices currently mapped by π in $\mathcal{X}$ and $\mathcal{Y}$, respectively. Since the configuration

$\mathcal{X}$ is connected, there is at least one pair of edges $\langle(c_1, i_1), (c_2, i_2)\rangle$ in $\mathcal{X}$ such that $c_1 \in \mathsf{cindex}(\pi, \mathcal{X})$ and $c_2 \notin \mathsf{cindex}(\pi, \mathcal{X})$ or $c_1 \notin \mathsf{cindex}(\pi, \mathcal{X})$ and $c_2 \in \mathsf{cindex}(\pi, \mathcal{X})$. Out of these pairs we choose the one whose permuted element is the smallest. Let us assume that this pair is $\langle(c_1, i_1), (c_2, i_2)\rangle$ and $c_1 \in \mathsf{cindex}(\pi, \mathcal{X})$ (the other case where $c_2 \in \mathsf{cindex}(\pi, \mathcal{X})$ can be treated in a similar fashion). We extend π to map c_2 to c', where $c' = \max(\mathsf{cindex}(\pi, \mathcal{Y})) + 1$, in order to produce the smallest possible $\mathcal{Y}$. The corresponding offset is chosen such that it makes the index i' in $\pi(c_2, i_2) = (c', i')$ as small as possible. After updating $\mathsf{cindex}(\pi, \mathcal{X}) = \{c_2\} \cup \mathsf{cindex}(\pi, \mathcal{X})$ and $\mathsf{cindex}(\pi, \mathcal{Y}) = \{c'\} \cup \mathsf{cindex}(\pi, \mathcal{Y})$, the permutation π can be extended in this way as long as $\mathcal{X}$ contains pairs in which one of the cycle identifiers has already been permuted and the other has not. If at any point $\mathcal{X}$ contains only pairs where both cycle indices are either permuted or not permuted, and $|\mathsf{cindex}(\pi, \mathcal{Y})| \neq c_{\mathsf{max}} + 1$, where c_{max} is the largest cycle number appearing in $\mathcal{X}$, then none of the cycles in the range of π are attached to cycles outside its range. This would mean that the graph corresponding to the configuration consists of at least two disjoint components, which contradicts the fact the augmentation algorithm we use only produces connected configurations.

Example 10. *Let $\mathcal{X}$ be configuration*

$$\mathcal{X} = (\langle(0,0),(1,1)\rangle, \langle(0,1),(1,2)\rangle, \langle(0,2),(0,7)\rangle, \langle(1,0),(1,3)\rangle)$$

Let us simulate body of the inner loop in Algorithm 2 with values $c = 1$ and $a = 0$. Now $\mathsf{cindex}(\pi, \mathcal{X}) = \{1\}$ and $\mathsf{cindex}(\pi, \mathcal{Y}) = \{0\}$ with π mapping edge $(1, i)$ to $(0, i)$ for all $i \in \{0, \ldots, 7\}$. Since $0 \notin \mathsf{cindex}(\pi, \mathcal{X})$ the edges $(0, i)$ are outside the domain of π. Configuration $\mathcal{X}$ becomes

$$(\underbrace{\langle\underline{(0,0)},(0,1)\rangle, \langle\underline{(0,1)},(0,2)\rangle}_{\textit{pairs with only one edge mapped by } \pi}, \langle\underline{(0,2)},\underline{(0,7)}\rangle\langle(0,0),(0,3)\rangle)$$

After applying π on the edges with the underlined edges being outside the domain of π. Now to find out how to π should map edges of cycle 0 we need to consider the pairs that have one edge in the domain of π and the other outside the domain, i.e., the first two pairs. Out of these two pairs we choose the one whose edge belonging to the domain of π is the smallest, namely $\langle\underline{(0,0)},(0,1)\rangle$.

Observe that the least cycle available is 1 since $\mathsf{cindex}(\pi, \mathcal{Y}) = \{0\}$ and mapping cycle 0 to any value > 1 would necessarily result in a larger configuration than mapping cycle 0 to 1. Compare, e.g., pairs $\langle(0,1),(1,0)\rangle$ and $\langle(0,1),(2,0)\rangle$. Additionally the best offset o_0 is 0 since any other offset would result in edge $(0,0)$ to be mapped to an edge greater than $(1,0)$. We thus extend π to map cycle 0 to 1 with offset 0.

Applying the updated π on the edges of $\mathcal{X}$ we get

$$(\langle(1,0),(0,1)\rangle, \langle(1,1),(0,2)\rangle, \langle(1,2),(1,7)\rangle, \langle(0,0),(0,3)\rangle)$$

which after sorting yields

$$\mathcal{Y} = \pi(\mathcal{X}) = (\langle(0,0),(0,3)\rangle, \langle(0,1),(1,0)\rangle, \langle(0,2),(2,1)\rangle, \langle(1,2),(1,7)\rangle)$$

Now the algorithm would terminate returning false *since $\mathcal{Y} < \mathcal{X}$, i.e., a permutation yielding a smaller configuration was found.*

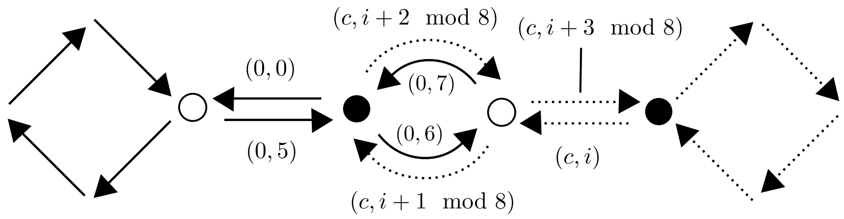


Figure 4.5: 8-cycles passing a double edge.

4.6 Optimizations

We have already noted some optimizations such as the ones regarding augmenting, which reduce the number of augmentations of each configuration by ruling out ones that are guaranteed to yield invalid or non-canonical configurations. Since using these optimizations guarantees that every augmentation of a valid configuration is bipartite and connected we may dispose of checking these two properties in the validity checking procedure. However, the validity checking procedure is still required to ensure extendability into a configuration with underlying 3-regular graph.

In addition to these optimizations we discovered that configurations of maximum length with a certain prefix can never be canonical. To elaborate, let $N = 6(g-1)$ for an arbitrary natural number $g > 1$ and consider configurations built out of N bipartite, directed 8-cycles. Now since each configuration consists of pairs of edge identifiers such that each edge appears at most once, and there are $8N$ edge identifiers the maximum length of a configuration must be $4N$. In a configuration of length $4N$ each edge identifier must then appear exactly once.

Theorem 9. *Assume that* $\mathcal{X} = (x_0, \ldots, x_{4N-1})$ *is a configuration of length* $4N$ *and that* $x_0 = \langle (0,0), (0,5) \rangle$. *Then* $\mathcal{X}$ *is not canonical.*

Proof. Since $\mathcal{X}$ is a configuration of maximum length we know that every edge identifier from the N 8-cycles appears exactly once in $\mathcal{X}$. This means that every directed edge e has been paired up with some other directed edge e' in $\mathcal{X}$. Additionally since cycle 0 has a self-attachment, namely $\langle (0,0), (0,5) \rangle$, we know that it traverses a double edge. This is because edges $(0,6)$ and $(0,7)$ cannot be attached to each other (see Figure 4.6) meaning that they must be paired up with edges from cycles other than 0. However, edges $(0,6)$ and $(0,7)$ can only be paired up with consecutive edges from the same cycle as depicted in Figure 4.5 since any other arrangement would not meet the requirement of 3-regularity. We therefore know that there exists an $i \in \{0, \ldots, 7\}$ such that pair $\langle (c,i), (c, i+3 \mod 8) \rangle$ is in $\mathcal{X}$ where c is the cycle whose edges are attached to $(0,6)$ and $(0,7)$, see Figure 4.5.

Now to show that $\mathcal{X}$ is not the minimal representative of its equivalence class we construct a permutation π mapping $\mathcal{X}$ to $\mathcal{Y} = (y_0, \ldots, y_{4N-1})$ for which $y_0 = \langle (0,0), (0,3) \rangle$. Let $\pi \in S_N \times C_4^N$ be the permutation swapping cycles 0 and c while keeping other cycles as they are, and let the offset of cycle c be $o_c = -i$ with all other offsets being zero, i.e., $o_j = 0$ for all $j \in \{0, \ldots, N-1\} \setminus \{c\}$. Combining this with the fact that $\langle (c,i), (c, i+3 \mod 8) \rangle \in \mathcal{X}$ we know that $\pi(\langle (c,i), (c, i+3 \mod 8) \rangle) = \langle (0,0), (0,3) \rangle \in \mathcal{Y}$. Since $\langle (0,0), (0,3) \rangle$ is the smallest possible pair it is clear that $y_0 = \langle (0,0), (0,3) \rangle$. And now $\mathcal{Y} < \mathcal{X}$ because $y_0 < x_0$, and therefore $\mathcal{X}$ is not canonical. $\square$

Using Theorem 9 we may then state that there are only two pairs that a canonical configuration of length $4N$ may have as its first element.

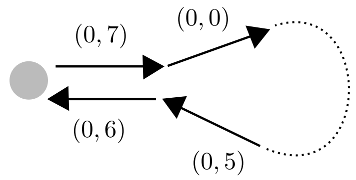


Figure 4.6: Pair $\langle(0,6),(0,7)\rangle$ is cannot appear in any $\mathcal{X}$ since this would result in a *blocked* node of degree 1. In fact no pair $\langle(0,i),(0,i+1 \mod 8)\rangle$ for $i \in I$ is allowed in any configuration.

Corollary 4. *Let* $\mathcal{X} = (x_0, \ldots, x_{4N-1})$ *be a canonical configuration of length* $4N$*. Now* x_0 *must be either* $\langle(0,0),(0,3)\rangle$ *or* $\langle(0,0),(1,1)\rangle$*.*

Proof. Denote $x_0 = \langle(c,i),(c',i')\rangle$. Clearly (c,i) must be the least edge identifier $(0,0)$ since if it were any other the pair would not be the first in $\mathcal{X}$. Now since i is even we know that i' must be odd, and the case $c' > 1$ is impossible since we could apply a permutation mapping c' to 1 yielding a smaller configuration. We additionally know that cases $(c',i') \in \{(0,1),(0,7)\}$ are impossible since that would lead to an invalid configuration due to creating a node of degree 1, see Figure 4.6. The case $(c',i') = (0,5)$ is ruled out by Theorem 9. Cases $(c',i') \in \{(1,3),(1,5),(1,7)\}$, on the other hand, would contradict the canonicity of $\mathcal{X}$ since we could permute such that (c',i') maps to $(1,1)$. Now the only remaining choices for (c',i') are $\langle(0,0),(0,3)\rangle$ and $\langle(0,0),(1,1)\rangle$. $\square$

The cases $(c',i') \in \{(1,3),(1,5),(1,7)\}$ of the previous corollary are ruled out by the algorithm already when considering a configuration $\mathcal{X} = (\langle(0,0),(c',i')\rangle)$ of length 1. This is because the canonicity checking procedure will discover the permutation π mapping (c',i') to $(1,1)$ and $\mathcal{X}$ will be ruled out as non-canonical.

The case $\mathcal{X} = (\langle(0,0),(0,5)\rangle)$, however, is not immediately ruled out as non-canonical. This is because there exists no permutation mapping cycle 0 to itself such that pair $\langle(0,0),(0,5)\rangle$ is mapped to $\langle(0,0),(0,3)\rangle$. Any configuration starting with $\langle(0,0),(0,5)\rangle$ is eventually ruled out for not being canonical, but this requires knowledge of the other cycle with self-attachment attached to cycle 0 as in the proof of Theorem 9. Therefore the algorithm may dispose of such a configuration as non-canonical only after the respective pair $\langle(c,i),(c,i+3 \mod 8)\rangle$ has been added to the configuration, and this may require building up a configuration of considerable length. Due to Theorem 9, however, we may simply skip configurations starting with $\langle(0,0),(0,5)\rangle$.

Chapter 5

Experiments and Results

In this chapter we report on results obtained by employing different combinations of orderly generation (recall Chapter 4) and the SAT encodings for enumerating cycleset decompositions and labeling graph-cycleset pairs (recall Sections 3.2 and 3.3, respectively). In particular, we confirm the results earlier reported in [56] for genus $g = 2$ using two semi-independent ways. Furthermore, we exhaustively treat the cases of $g = 3$ and $g = 4$, altogether ruling out 4 further groups out of the 23 T_i's. We report on the runtime distribution of employing the MiniSAT solver [94] through the PySAT interface [95] for finding cyclesets and labelings. All experiments were run on computing nodes with Xeon E5-2680 v4 2.4-GHz processors and 256-GB RAM under CentOS 7. Our implementation, empirical data and witness graphs found are available via `https://bitbucket.org/coreo-group/periodic-apartments/`.

In the following, we will refer by G+SAT^2 to the approach consisting of

(i) generating $\mathsf{base}(g)$, i.e., all connected, bipartite, and 3-regular graphs with $16(g-1)$ nodes and $24(g-1)$ edges (for a given genus g) using off-the-shelf tool Multigraph [76];

(ii) using the SAT encoding of Section 3.2 to enumerate the cyclesets of the graphs in (i); and

(iii) using the SAT encoding of Section 3.3 to determine the existence of a labeling for the graph-cycleset pairs from (ii).

In contrast, we will refer by OG+labelSAT to the approach consisting of

(i') generating the graph-cycleset pairs directly with the orderly approach of Chapter 4, and

(ii') Checking the existence of a labeling for each pair using the encoding of Section 3.3.

Table 5.1: Groups ruled out at genus g with those not ruled out at smaller value of g in bold.

Approach	**Genus 2**	**Genus 3**	**Genus 4**
G+SAT^2	T_1, T_2, T_7, T_9, T_{18}		
OG+labelSAT	T_1, T_2, T_7, T_9, T_{18}	T_1, T_2, $\boldsymbol{T_6}$, T_7, T_9, $\boldsymbol{T_{13}}$, $\boldsymbol{T_{16}}$, T_{18}	T_1, T_2, T_7, T_9, $\boldsymbol{T_{15}}$, T_{18}

5.1 Confirmation of Earlier Results for Genus 2

Kangaslampi and Vdovina exhaustively treated the genus 2 case [56]. Their approach consisted of

(i) generating all connected, bipartite, 3-regular graphs with 16 nodes and 24 edges while treating simple and non-simple graphs separately;

(ii) for each of these 773 graphs a depth-first search for determining the sets of 6 8-cycles, and

(iii) specialized depth-first search over each of the graph-cycleset pairs to determine if the pair admits a labeling.

As reported in [56], this approach does not scale beyond $g = 2$. Both our approaches differ from the one used by Kangaslampi and Vdovina. The G+SAT^2 approach differs in terms of using SAT solvers for enumerating the possible cycleset decompositions (see Section 3.2) and checking each graph-cycleset pair for a valid labeling (see Section 3.3). The OG+labelSAT approach, on the other hand, generates directly a stricter set of graphs; $\mathsf{cycles}(g)$ instead of $\mathsf{base}(g)$. OG+labelSAT also employs a SAT solver for checking the existence of labelings. Both our approaches are thus independent of the one by Kangaslampi and Vdovina. We, however, call our two approaches *semi-independent* since they share the last part in which graph-cycleset pairs are checked for valid labelings.

Using both G+SAT^2 and OG+labelSAT we exhaustively treated the case of genus $g = 2$. The results obtained with these approaches were identical: both approaches found genus 2 periodic apartments for groups T_1, T_2, T_7, T_9 and T_{18} (see Table 5.1). These results agree perfectly with those reported in [56]. The genus 2 results have thus been reproduced three times by methods which are at least semi-independent.

5.2 New Results beyond Genus 2

As already mentioned, Kangaslampi and Vdovina were unable to scale their approach beyond genus 2. In contrast, our OG+labelSAT approach, using straightforward parallelization, allowed for an efficient exhaustive analysis of genera 3 and 4. As a result, we are able to rule out four more groups: T_6, T_{13}, T_{15} and T_{16} (the groups in bold in Table 5.1). We provide concrete witnesses for each T_i and g for which $\mathsf{labels}(T_i, g)$ is nonempty. Each concrete witness is a graph in $\mathsf{labels}(T_i, g)$ proving the nonemptiness of the set, and the membership of each such graph in $\mathsf{labels}(T_i, g)$ is fairly straightforward to check. See Section 5.5 for a discussion regarding the correctness and reliability of the results. Examples of concrete witnesses for the four new groups T_6, T_{13}, T_{15}, and T_{16} are provided in Appendix B. For an exhaustive listing of the witness graphs found, see the website `https://bitbucket.org/coreo-group/periodic-apartments/`

For groups T_6, T_{13} and T_{16} we discovered genus–3 graphs whereas for T_{15} we discovered genus–4 graphs. Overall we may conclude that groups T_1, T_2, T_6, T_7, T_9, T_{13}, T_{15}, T_{16} and T_{18} are the only groups out of the 23 T_i's that have a periodic apartment of genus ≤ 4. Since the existence of a periodic apartment implies the existence of a surface subgroup (see Theorem 1 and Corollary 1) these groups are thus ruled out as possible counterexamples to Gromov subgroup conjecture.

Table 5.2: Statistics for different steps of the approaches.

Approach	**Genus**	**Orientable graphs**	**Labelable graphs**	**Pairs**	**Hits**
G+SAT^2	2	12	4	274	152
OG+labelSAT	2	12	4	84	15
OG+labelSAT	3	1399	26	5 872	67
OG+labelSAT	4	–	127	6 125 906	491

5.3 Numerical Data

Table 5.2 gives more detailed statistics on the different steps on the approaches. The columns **Orientable graphs** and **Labelable graphs** show the number of graphs admitting a cycleset and the number of graphs admitting a valid labeling with some group T_i. The column **Pairs**, on the other hand, shows the number of graph-cycleset pairs for each approach, and the **Hits** column shows the number of graph-cycleset pairs admitting a labeling with some group T_i. The discrepancies for the genus 2 case in columns **Pairs** and **Hits** results from the fact that our orderly generation algorithm used in the OG+labelSAT approach achieved stronger symmetry breaking than the cycleset enumeration via a SAT encoding in G+SAT^2. Specifically, some cyclesets produced in G+SAT^2 using the encoding of Section 3.2 are the same modulo an automorphism of the graph in question. Note that the stronger symmetry breaking in OG+labelSAT results in a noticeably smaller average number of cyclesets per graph. The numbers of graphs for which cyclesets exists (column **Orientable graphs**) and which admit a valid labeling (**Labelable graphs**), on the other hand, are the same for G+SAT^2 and OG+labelSAT (as should be).

Table 5.3 shows the number of distinct graph-cycleset pairs that admit a labeling using each T_i for different genera. The groups missing from the table do not have any labeling-admitting graph-cycleset pairs for genus ≤ 4. Observe that groups T_1, T_2, T_7, T_9 and T_{18} have graph-cycleset pairs for genus values $g \in \{2, 3, 4\}$ with the number increasing as genus g increases. Groups T_6, T_{13} and T_{16}, on the other hand, have labelable graph-cycleset pairs only for genus 3 and their number is very low. Notice also that group T_{15} has a similarly low number of valid graph-cycleset pairs at genus 4.

Table 5.4 shows the sizes of $\mathsf{labels}(T_i, g)$, i.e., the numbers of non-isomorphic graphs of genus g admitting a labeling with T_i. A first observation is that most of these numbers are lower than the corresponding ones in Table 5.3 indicating that some graphs have multiple cyclesets admitting a valid labeling for various T_i's.

From this we can conclude using Corollary 1 that groups T_1, T_2, T_7, T_9, and T_{18} have surface subgroups of genera 2, 3, and 4. Groups T_6, T_{13}, and T_{16}, however, have surface subgroups of genus 3, but no surface subgroups (arising from periodic apartments) of genera 2 and 4. Similarly group T_{15} has a surface subgroup of genus 4 but no surface subgroups of genera 2 or 3 arising from periodic apartments.

5.4 Performance

The SAT-based labeling phase of both approaches was quite efficient for genus 2, with cumulative runtimes of 420 seconds for G+SAT^2 and 143 seconds for OG+labelSAT. The SAT-based cycleset generation phase of G+SAT^2 over the 773 graphs generated

Table 5.3: The numbers of graph-cycleset pairs admitting a labeling with each group T_i.

	T_1	T_2	T_6	T_7	T_9	T_{13}	T_{15}	T_{16}	T_{18}
genus 2	9	3	0	3	3	0	0	0	6
genus 3	16	5	4	6	5	1	0	9	36
genus 4	133	22	0	35	31	0	6	0	348

Table 5.4: The numbers of non-isomorphic graphs admitting a labeling with each group T_i.

	T_1	T_2	T_6	T_7	T_9	T_{13}	T_{15}	T_{16}	T_{18}
genus 2	2	1	0	1	1	0	0	0	2
genus 3	6	3	2	3	2	1	0	7	10
genus 4	40	12	0	15	12	0	2	0	82

using Multigraph (phase (i)) took a total of 348 seconds, while the orderly generation phase of OG+labelSAT took 3 seconds, which suggested that OG+labelSAT would scale better of the two to larger genera. The better scaling of OG+labelSAT to higher genera was supported by the numbers of graphs / graph-cycleset pairs generated at genus 3.

While Multigraph (phase (i) of G+SAT^2) would generate 13 703 003 409 graphs at genus 3, orderly generation at genus 3 resulted in 5872 graph-cycleset pairs, out of which 67 admit a labeling; see Table 5.2. For genus 4 we generated 6 125 906 graph-cycleset pairs, out of which 491 admit a labeling. At genus 3, orderly generation took approximately 14 hours and the labeling phase less than 8 hours. Orderly generation of genus 4 configurations took 883 500 hours, and checking them for labeling took 7 241 hours. Figure 5.1 shows the total runtime of the labeling phase per graph-cycleset pair over the 23 T_i's for genus 3 (left) and genus 4 (right). The number plotted in the graphs is thus the sum of the runtimes of 23 SAT calls; one for each group T_i. The shape of the curve suggests that large majority of the graph-cycleset pairs required little time for the labeling phase and a small fraction required much longer. The median runtimes of the labeling phase at genus 3 and 4 are approximately 2 and 3 seconds, respectively. It is also worth stating that each *individual* SAT call took less than 2 seconds at genus 3 and 12 seconds at genus 4.

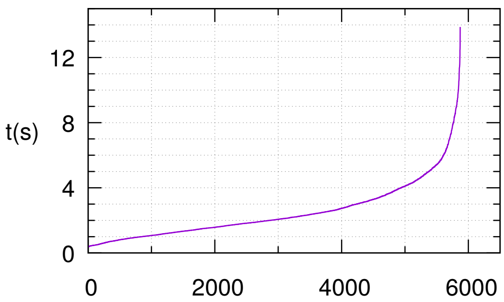
5.5 On Correctness

We may draw two rather different conclusions from the experiments and their results as described previously:

(i) the existence of periodic apartments for certain groups T_i, and

(ii) the *non-existence* of periodic apartments for other T_i.

Firstly we claim that groups T_1, T_2, T_6, T_7, T_9, T_{13}, T_{16} and T_{18} have periodic apartments of genus 3, and this claim is evidenced by graphs in $\mathsf{labels}(T_i, 3)$ for $i \in \{1, 2, 6, 7, 9, 13, 16, 18\}$. Similarly we claim that groups T_1, T_2, T_7, T_9, T_{15} and T_{18} have a periodic apartment of genus 4 witnessed by graphs in $\mathsf{labels}(T_{15}, i)$ for $i \in \{1, 2, 7, 9, 15, 18\}$. Examples of these graphs for T_6, T_{13}, T_{15} and T_{16} are shown in Appendix B. Secondly, we claim that no group other than T_1, T_2, T_6, T_7, T_9, T_{13}, T_{16}



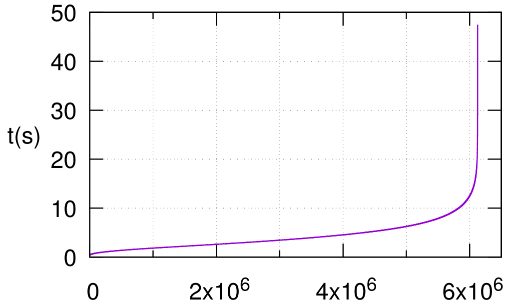


Figure 5.1: Runtime distribution of the labeling phase of OG+labelSAT for genus 3 (left) and genus 4 (right). The total runtime of 23 SAT calls (one for each T_i) per graph-cycleset pair is plotted.

or T_{18} have a periodic apartment of genus 3, and similarly that no group other than T_1, T_2, T_7, T_9, T_{15} or T_{18} have a periodic apartment of genus 4.

The validity of the claims of type (i) hinges on the correctness of Theorem 1 and whether it holds that $\mathsf{labels}(T_i, g) \neq \emptyset$. Theorem 1 has been shown in [56], and the non-emptiness of $\mathsf{labels}(T_i, g)$ is witnessed by concrete graphs, which can be manually checked to be contained in $\mathsf{labels}(T_i, g)$. Therefore, after the concrete witnesses we have produced have been validated, claims (i) do not depend on the correctness of our SAT encodings (Sections 3.2 and 3.3) or the orderly generation algorithms (Chapter 4).

The claims of type (ii), however, depend critically on the correctness of the SAT encoding of Section 3.3 as well as the orderly generation algorithm of Chapter 4. The reason is that claims (ii), after applying Theorem 1, reduce to showing that $\mathsf{labels}(T_i, g)$ is empty, and our approach essentially enumerates $\mathsf{cycles}(g) \supseteq \mathsf{labels}(T_i, g)$ and then checks whether each $G \in \mathsf{cycles}(g)$ is contained in $\mathsf{labels}(T_i, g)$. The main difference is thus that claims (i) are existential claims ("There is a graph with properties ...") which can be validated by checking the concrete witnesses produced whereas (ii) are universal claims ("Every graph fails to satisfy ...").

Necessary and sufficient conditions for the correctness of orderly generation algorithms are outlined by Read in [22]. Using these conditions it is possible to prove the correctness of our orderly generation algorithm, but this does not suffice since the implementation itself may contain bugs. The implementation and algorithm could of course be formally verified, but we have refrained from doing so in this work.

When it comes to the SAT encoding we must consider whether the encoding works as intended as well as the correctness of the SAT solver in use. Any correctness issues with SAT solvers, however, can be remedied using existing techniques. One option would be to use formally verified SAT solvers [104, 105, 106]. While formally verified solvers guarantee the correctness of their output their performance tends to be lower than the best-available solvers. Another option would be to use a proof producing SAT solver, i.e., one that produces either a model or a resolution refution of the input formula [107, 108, 109, 110, 111]. The resolution refutation of an unsatisfiable formula can then be checked independently of the used solver. The relatively low overhead of proof logging and verification allows it to be employed even for large proofs such as the proof of Boolean Pythagorean triples [2].

On another note, the heuristic reliability of some results can also be enhanced by reproduction of the same results via independent means. We have, for example, produced the results for genus 2 using two semi-independent methods (G+SAT^2 and OG+labelSAT), and these results agree perfectly with the ones produced by Kangaslampi and Vdovina [56]. While this may make the results easier to trust it is no formal proof.

All in all we consider the positive results, i.e., the discovered periodic apartments, to be highly reliable, especially the genus–2 results which have been reproduced several times. The negative results, i.e., the claims of the non-existence of periodic apartments for certain T_i, should be trusted under the assumption that our SAT encodings and orderly generation algorithm are correct and correctly implemented.

Chapter 6

Conclusions

We presented a computational study of the applicability of combinations of SAT solving and orderly generation to a problem arising from geometric group theory, dealing in particular with determining whether one of 23 specific groups earlier put forth by Kangaslampi and Vdovina [56] would serve as a counterexample to the famous subgroup conjecture of Gromov. While earlier computational treatment of this problem setting was restricted to genus 2 [56], we showed that a combination of SAT solving and orderly generation allows for significantly scaling up (by several orders of magnitude) to genera 3 and 4. As a result, we provided an independent confirmation of the earlier results for genus 2 [56], and ruled out four more groups out of the 23 as counterexamples to Gromov's subgroup conjecture by exhaustively treating genera 3 and 4.

While we utilized orderly generation to produce $\mathsf{cycles}(g)$ and SAT solvers to check which of the generated inputs belong to $\mathsf{labels}(T_i, g)$, other approaches could have been taken as well. We considered using SAT solvers to generate the cycleset-admitting graphs $\mathsf{cycles}(g)$ or the periodic apartments $\mathsf{labels}(T_i, g)$ directly. The problem with these approaches seemed to be the existence of numerous symmetries making it difficult to formulate a performant encoding. Utilizing efficient symmetry-breaking we could use SAT solvers to directly generate the graphs in $\mathsf{labels}(T_i, g)$ and this is indeed a possible avenue of further research.

Another possibility would be to modify our orderly generation algorithm to generate $\mathsf{labels}(T_i, g)$ directly. This would require us to develop an efficient canonicity checking algorithm and would probably require several problem-specific optimizations like our generator for $\mathsf{cycles}(g)$.

The problem of finding periodic apartments seems to be connected to edge-matching puzzles [112, 113, 114] as well and investigating this connection is another possible line of further research. Efficient computational methods used to solve edge-matching puzzles could be helpful in finding periodic apartments since the problem boils down to matching a specific number of triangles whose edges are labeled. In our case, however, the tesselation of these triangles is not in the 2-dimensional plane.

The *cycle double cover conjecture* [115], which is a long-standing open problem in graph theory, is also connected to the problem of periodic apartments considered in this work. The conjecture states that "each bridgeless graph contains 2-cover consisting entire of cycles". The computational tools used in this work, e.g., the encoding introduced in Section 3.2, could be modified to study the cycle double cover conjecture empirically. Computational methods have been used in studying this conjecture [116] but no counterexamples have been found.

Yet another possibility opened up by the results of our experiments is that of manual inspection of the found graphs, cyclesets, and labelings to try and derive some insight into the problem of finding periodic apartments. Many of the witness graphs found seem symmetric even by visual examination, see the graph in Figure B.2 for example.

Acknowledgments

I finished this thesis while working in the Constraint Reasoning and Optimization group (CoReO) led by Associate Professor Matti Järvisalo at University of Helsinki. I am indebted to my colleagues in the group and staff of the Department of Computer Science, especially the IT for Science group. To my advisors, Associate Professor Matti Järvisalo and Docent Emilia Oikarinen, I offer my deepest gratitude. It was their encouragement, patience and invaluable advice that made this work possible. I also offer my thanks to Doctor Riikka Kangaslampi for insightful discussions on the problem and to Doctor Markus Meringer and Professor Gunnar Brinkmann for correspondence on orderly generation and Multigraph. I also thank my partners, family and friends for their love and support during the writing of this thesis.

This work was financially supported by Academy of Finland (grants 312662 and 322869). Computational resources were provided by Finnish Grid and Cloud Infrastructure (FCGI) [117].

Bibliography

[1] B. Konev and A. Lisitsa, "Computer-aided proof of Erdős discrepancy properties," *Artificial Intelligence*, vol. 224, pp. 103–118, 2015.

[2] M. Heule, O. Kullmann, and V. Marek, "Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer," in *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings* (N. Creignou and D. Le Berre, eds.), vol. 9710 of *Lecture Notes in Computer Science*, pp. 228–245, Springer, 2016.

[3] M. Heule, "Schur number five," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (S. McIlraith and K. Weinberger, eds.), pp. 6598–6606, AAAI Press, 2018.

[4] F. Brandl, F. Brandt, M. Eberl, and C. Geist, "Proving the incompatibility of efficiency and strategyproofness via SMT solving," *Journal of the ACM*, vol. 65, no. 2, pp. 6:1–6:28, 2018.

[5] J. Brakensiek, M. Heule, and J. Mackey, "The resolution of Keller's conjecture," *CoRR*, vol. abs/1910.03740, 2019.

[6] Z.-P. Xu, J.-L. Chen, and O. Gühne, "Proof of the Peres conjecture for contextuality," *Physical Review Letters*, vol. 124, 2020.

[7] P. Herwig, M. Heule, M. van Lambalgen, and H. van Maaren, "A new method to construct lower bounds for Van der Waerden numbers," *The Electronical Journal of Combinatorics*, vol. 14, no. 1, 2007.

[8] F. Brandt, C. Geist, and D. Peters, "Optimal bounds for the no-show paradox via SAT solving," *Mathematical Social Sciences*, vol. 90, pp. 18–27, 2017.

[9] N. Francetic, S. Herke, B. McKay, and I. Wanless, "On Ryser's conjecture for linear intersecting multipartite hypergraphs," *European Journal of Combinatorics*, vol. 61, pp. 91–105, 2017.

[10] F. Brandt, P. Harrenstein, and H. Seedig, "Minimal extending sets in tournaments," *Mathematical Social Sciences*, vol. 87, pp. 55–63, 2017.

[11] J. Goedgebeur, K. Ozeki, N. van Cleemput, and G. Wiener, "On the minimum leaf number of cubic graphs," *Discrete Mathematics*, vol. 342, no. 11, pp. 3000–3005, 2019.

[12] N. Kaplan, S. Kimport, R. Lawrence, L. Peilen, and M. Weinreich, "Counting arcs in projective planes via Glynn's algorithm," *Journal of Geometry*, vol. 108, no. 3, pp. 1013–1029, 2017.

[13] O. Kullmann, "Green-Tao numbers and SAT," in *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings* (O. Strichman and S. Szeider, eds.), vol. 6175 of *Lecture Notes in Computer Science*, pp. 352–362, Springer, 2010.

[14] C. Geist and U. Endriss, "Automated search for impossibility theorems in social choice theory: Ranking sets of objects," *Journal of Artificial Intelligence Research*, vol. 40, pp. 143–174, 2011.

[15] F. Brandt and C. Geist, "Finding strategyproof social choice functions via SAT solving," *Journal of Artificial Intelligence Research*, vol. 55, pp. 565–602, 2016.

[16] B. Klocker, H. Fleischner, and G. Raidl, "A SAT approach for finding sup-transition-minors," in *Learning and Intelligent Optimization - 13th International Conference, LION 13, Chania, Crete, Greece, May 27-31, 2019, Revised Selected Papers* (N. Matsatsinis, Y. Marinakis, and P. Pardalos, eds.), vol. 11968 of *Lecture Notes in Computer Science*, pp. 325–341, Springer, 2019.

[17] L. Finschi and K. Fukuda, "Complete combinatorial generation of small point configurations and hyperplane arrangements," in *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pp. 97–100, 2001.

[18] C. Desrosiers, P. Galinier, P. Hansen, and A. Hertz, "Automated generation of conjectures on forbidden subgraph characterization," *Discrete Applied Mathematics*, vol. 162, pp. 177–194, 2014.

[19] P. Östergård, P. Lampio, and F. Szöllősi, "Orderly generation of Butson Hadamard matrices," *Mathematics of Computation*, vol. 89, no. 321, pp. 313–331, 2020.

[20] Y. Matsumoto, S. Moriyama, H. Imai, and D. Bremner, "Matroid enumeration for incidence geometry," *Discrete & Computational Geometry*, vol. 47, no. 1, pp. 17–43, 2012.

[21] A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds., *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[22] R. Read, "Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations," *Annals of Discrete Mathematics*, vol. 2, pp. 107–120, 1978.

[23] P. de la Harpe, *Topics in geometric group theory*. Chicago Lectures in Mathematics, University of Chicago Press, 2000.

[24] C. Druţu and M. Kapovich, *Geometric group theory*, vol. 63 of *Colloquium Publications*. American Mathematical Society, 2018.

[25] M. Gromov, "Hyperbolic groups," in *Essays in group theory*, pp. 75–263, Springer, 1987.

[26] M. Mitra, "Ending laminations for hyperbolic group extensions," *Geometric and Functional Analysis*, vol. 7, no. 2, p. 379, 1997.

[27] O. Kharlampovich and A. Myasnikov, "Hyperbolic groups and free constructions," *Transactions of the American Mathematical Society*, vol. 350, no. 2, pp. 571–613, 1998.

[28] B. Bowditch, "Cut points and canonical splittings of hyperbolic groups," *Acta mathematica*, vol. 180, no. 2, pp. 145–186, 1998.

[29] N. Brady, "Finite subgroups of hyperbolic groups," *International Journal of Algebra and Computation*, vol. 10, no. 4, pp. 399–406, 2000.

[30] M. Coornaert and G. Knieper, "An upper bound for the growth of conjugacy classes in torsion-free word hyperbolic groups," *International Journal of Algebra and Computation*, vol. 14, no. 4, pp. 395–401, 2004.

[31] J. Roe, "Hyperbolic groups have finite asymptotic dimension," *Proceedings of the American Mathematical Society*, vol. 133, no. 9, pp. 2489–2490, 2005.

[32] E. Jaligot, A. Muranov, and A. Neman, "Independence property and hyperbolic groups," *Bulletin of Symbolic Logic*, vol. 14, no. 1, pp. 88–98, 2008.

[33] D. Osin, "On the universal theory of torsion and lacunary hyperbolic groups," *Groups Complexity Cryptology*, vol. 1, no. 2, pp. 311–319, 2009.

[34] M. Deraux, J. Parker, and J. Paupert, "Census of the complex hyperbolic sporadic triangle groups," *Experimental Mathematics*, vol. 20, no. 4, pp. 467–486, 2011.

[35] A. Ol'shanskii, "Almost every group is hyperbolic," *International Journal of Algebra and Computation*, vol. 2, no. 1, pp. 1–18, 1992.

[36] F. Dahmani and V. Guirardel, "Foliations for solving equations in groups: free, virtually free, and hyperbolic groups," *Journal of Topology*, vol. 3, no. 2, pp. 343–404, 2010.

[37] F. Dahmani and V. Guirardel, "The isomorphism problem for all hyperbolic groups," *Geometric and Functional Analysis*, vol. 21, no. 2, pp. 223–300, 2011.

[38] J. Tits, *Buildings of spherical type and finite BN-pairs*, vol. 386. Springer-Verlag, 1974.

[39] J. Tits, "Structures et groupes de Weyl," in *Séminaire Bourbaki (1964/1965), Éxpose No. 288*, pp. 169–183, Société Mathématique de France, 1966.

[40] P. Abramenko and K. Brown, *Buildings: theory and applications*, vol. 248 of *Graduate Texts in Mathematics.* Springer Science & Business Media, 2008.

[41] J. Tits, "Sur la trialité et certains groupes qui s'en déduisent," *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, vol. 2, no. 1, pp. 14–60, 1959.

[42] A. Thomas, "Lattices in hyperbolic buildings," *Geometry, Topology, and Dynamics in Negative Curvature*, vol. 425, p. 345, 2016.

[43] T. Marquis, "On geodesic ray bundles in buildings," *Geometriae Dedicata*, vol. 202, pp. 27–43, 2019.

[44] E. Stark, "Topological rigidity fails for quotients of the Davis complex," *Proceedings of the American Mathematical Society*, vol. 146, no. 12, pp. 5357–5366, 2018.

[45] D. Constantine and J.-F. Lafont, "Marked length rigidity for Fuchsian buildings," *Ergodic Theory and Dynamical Systems*, vol. 39, no. 12, pp. 3262–3291, 2019.

[46] J. Bounds and X. Xie, "Quasi-isometric rigidity of a class of right-angled Coxeter groups," *Proceedings of the American Mathematical Society*, vol. 148, no. 2, pp. 553–568, 2020.

[47] C. Gordon, D. Long, and A. Reid, "Surface subgroups of Coxeter and Artin groups," *Journal of Pure and Applied Algebra*, vol. 189, no. 1, pp. 135 – 148, 2004.

[48] D. Calegari, "Surface subgroups from homology," *Geometry & Topology*, vol. 12, no. 4, pp. 1995–2007, 2008.

[49] C. Gordon and H. Wilton, "On surface subgroups of doubles of free groups," *Journal of the London Mathematical Society*, vol. 82, no. 1, pp. 17–31, 2010.

[50] D. Futer and A. Thomas, "Surface quotients of hyperbolic buildings," *International Mathematics Research Notices*, vol. 2012, no. 2, pp. 437–477, 2011.

[51] H. Wilton, "One-ended subgroups of graphs of free groups with cyclic edge groups," *Geometry & Topology*, vol. 16, no. 2, pp. 665–683, 2012.

[52] V. Markovic, "Criterion for Cannon's conjecture," *Geometric and Functional Analysis*, vol. 23, no. 3, pp. 1035–1061, 2013.

[53] S. Kim and S. Oum, "Hyperbolic surface subgroups of one-ended doubles of free groups," *Journal of Topology*, vol. 7, no. 4, pp. 927–947, 2014.

[54] D. Calegari and A. Walker, "Random groups contain surface subgroups," *Journal of the American Mathematical Society*, vol. 28, no. 2, pp. 383–419, 2015.

[55] H. Wilton, "Essential surfaces in graph pairs," *Journal of the American Mathematical Society*, vol. 31, no. 4, pp. 893–919, 2018.

[56] R. Kangaslampi and A. Vdovina, "Hyperbolic triangular buildings without periodic planes of genus 2," *Experimental Mathematics*, vol. 26, no. 1, pp. 54–61, 2017.

[57] R. Kangaslampi and A. Vdovina, "Cocompact actions on hyperbolic buildings," *International Journal of Algebra and Computation*, vol. 20, no. 4, pp. 591–603, 2010.

[58] A. Vdovina, "Combinatorial structure of some hyperbolic buildings," *Mathematische Zeitschrift*, vol. 241, no. 3, pp. 471–478, 2002.

[59] W. Ballmann and M. Brin, "Polygonal complexes and combinatorial group theory," *Geometriae Dedicata*, vol. 50, no. 2, pp. 165–191, 1994.

[60] W. Ballmann and M. Brin, "Orbihedra of nonpositive curvature," *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, vol. 82, pp. 169–209, 1995.

[61] D. Gaboriau and F. Paulin, "Sur les immeubles hyperboliques," *Geometriae Dedicata*, vol. 88, no. 1-3, pp. 153–197, 2001.

[62] L. Carbone, R. Kangaslampi, and A. Vdovina, "Groups acting simply transitively on vertex sets of hyperbolic triangular buildings," *LMS Journal of Computation and Mathematics*, vol. 15, pp. 101–112, 2012.

[63] S. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA* (M. Harrison, R. Banerji, and J. Ullman, eds.), pp. 151–158, ACM, 1971.

[64] M. Heule, M. Järvisalo, and M. Suda, "SAT competition 2018," *Journal of Satisfiability, Boolean Modelling and Computation*, vol. 11, no. 1, pp. 133–154, 2019.

[65] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings* (P. van Beek, ed.), vol. 3709 of *Lecture Notes in Computer Science*, pp. 827–831, Springer, 2005.

[66] A. van Gelder, "Another look at graph coloring via propositional satisfiability," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 230–243, 2008.

[67] T. Philipp and P. Steinke, "PBLib - A library for encoding pseudo-boolean constraints into CNF," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings* (M. Heule and S. Weaver, eds.), vol. 9340 of *Lecture Notes in Computer Science*, pp. 9–16, Springer, 2015.

[68] V. Nguyen, "SAT encodings of finite-CSP domains: A survey," in *Proceedings of the Eighth International Symposium on Information and Communication Technology, Nha Trang City, Viet Nam, December 7-8, 2017*, pp. 84–91, ACM, 2017.

[69] A. Boudane, S. Jabbour, B. Raddaoui, and L. Sais, "Efficient SAT-based encodings of conditional cardinality constraints," in *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018* (G. Barthe, G. Sutcliffe, and M. Veanes, eds.), vol. 57 of *EPiC Series in Computing*, pp. 181–195, EasyChair, 2018.

[70] C. Colbourn and R. Read, "Orderly algorithms for generating restricted classes of graphs," *Journal of Graph Theory*, vol. 3, no. 2, pp. 187–195, 1979.

[71] G. Brinkmann, "Fast generation of cubic graphs," *Journal of Graph Theory*, vol. 23, no. 2, pp. 139–149, 1996.

[72] G. Royle, "An orderly algorithm and some applications in finite geometry," *Discrete Mathematics*, vol. 185, no. 1-3, pp. 105–115, 1998.

[73] M. Meringer, "Fast generation of regular graphs and construction of cages," *Journal of Graph Theory*, vol. 30, no. 2, pp. 137–146, 1999.

[74] J. Goedgebeur, *Generation Algorithms for Mathematical and Chemical Problems.* PhD thesis, Ghent University, 2013.

[75] A. Kohnert and S. Kurz, "Integral point sets over $\mathbb{Z}_n^m$," *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2105–2117, 2009.

[76] G. Brinkmann, "Multigraph." personal communication, 2019.

[77] J. Savela, E. Oikarinen, and M. Järvisalo, "Finding periodic apartments via Boolean satisfiability and orderly generation," in *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (E. Albert and L. Kovacs, eds.), vol. 73 of *EPiC Series in Computing*, pp. 465–482, EasyChair, 2020.

[78] J. Harrison, *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press, 2009.

[79] M. Ben-Ari, *Mathematical Logic for Computer Science, 3rd Edition.* Springer, 2012.

[80] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning 2*, pp. 466–483, Springer-Verlag, 1983.

[81] D. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *Journal of Symbolic Computation*, vol. 2, no. 3, pp. 293–304, 1986.

[82] J. Marques-Silva and K. Sakallah, "GRASP - a new search algorithm for satisfiability," in *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996* (R. Rutenbar and R. Otten, eds.), pp. 220–227, IEEE Computer Society / ACM, 1996.

[83] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[84] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pp. 530–535, ACM, 2001.

[85] C. Gomes, B. Selman, and H. Kautz, "Boosting combinatorial search through randomization," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA* (J. Mostow and C. Rich, eds.), pp. 431–437, AAAI Press / The MIT Press, 1998.

[86] E. Goldberg and Y. Novikov, "Berkmin: A fast and robust sat-solver," *Discrete Applied Mathematics*, vol. 155, no. 12, pp. 1549–1561, 2007.

[87] L. Ryan, *Efficient algorithms for clause-learning SAT solvers.* Master's thesis, Simon Fraser University, 2004.

[88] R. Bayardo and R. Schrag, “Using CSP look-back techniques to solve real-world SAT instances,” in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA* (B. Kuipers and B. Webber, eds.), pp. 203–208, AAAI Press / The MIT Press, 1997.

[89] M. Lewis, T. Schubert, and B. Becker, “Speedup techniques utilized in modern SAT solvers,” in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings* (F. Bacchus and T. Walsh, eds.), vol. 3569 of *Lecture Notes in Computer Science*, pp. 437–443, Springer, 2005.

[90] J. Crawford, M. Ginsberg, E. Luks, and A. Roy, “Symmetry-breaking predicates for search problems,” in *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR’96), Cambridge, Massachusetts, USA, November 5-8, 1996* (L. Aiello, J. Doyle, and S. Shapiro, eds.), pp. 148–159, Morgan Kaufmann, 1996.

[91] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, “Solving difficult SAT instances in the presence of symmetry,” in *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, pp. 731–736, ACM, 2002.

[92] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, “Solving difficult instances of Boolean satisfiability in the presence of symmetry,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1117–1137, 2003.

[93] F. Aloul, I. Markov, and K. Sakallah, “Shatter: efficient symmetry-breaking for boolean satisfiability,” in *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pp. 836–839, ACM, 2003.

[94] N. Eén and N. Sörensson, “An extensible SAT-solver,” in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers* (E. Giunchiglia and A. Tacchella, eds.), vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518, Springer, 2003.

[95] A. Ignatiev, A. Morgado, and J. Marques-Silva, “PySAT: A Python toolkit for prototyping with SAT oracles,” in *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings* (O. Beyersdorff and C. Wintersteiger, eds.), Lecture Notes in Computer Science, pp. 428–437, Springer, 2018.

[96] O. Grumberg, A. Schuster, and A. Yadgar, “Memory efficient all-solutions SAT solver and its application for reachability analysis,” in *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings* (A. Hu and A. Martin, eds.), vol. 3312 of *Lecture Notes in Computer Science*, pp. 275–289, Springer, 2004.

[97] Y. Yu, P. Subramanyan, N. Tsiskaridze, and S. Malik, "All-SAT using minimal blocking clauses," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, Mumbai, India, January 5-9, 2014*, pp. 86–91, IEEE Computer Society, 2014.

[98] S. Jabbour, J. Lonlac, L. Sais, and Y. Salhi, "Extending modern SAT solvers for models enumeration," in *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014, Redwood City, CA, USA, August 13-15, 2014* (J. Joshi, E. Bertino, B. Thuraisingham, and L. Liu, eds.), pp. 803–810, IEEE Computer Society, 2014.

[99] T. Toda and T. Soh, "Implementing efficient all solutions SAT solvers," *ACM Journal of Experimental Algorithmics*, vol. 21, no. 1, pp. 1.12:1–1.12:44, 2016.

[100] T. Toda and K. Tsuda, "BDD construction for all solutions SAT and efficient caching mechanism," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015* (R. Wainwright, J. Corchado, A. Bechini, and J. Hong, eds.), pp. 1880–1886, ACM, 2015.

[101] W. Zhao and W. Wu, "ASIG: an all-solution SAT solver for CNF formulas," in *11th International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2009, Huangshan, China, August 19-21, 2009*, pp. 508–513, IEEE, 2009.

[102] G. Brinkmann, "Minibaum webpage." http://caagt.ugent.be/minibaum/. Accessed: 2020-01-24.

[103] E. Luks, "Isomorphism of graphs of bounded valence can be tested in polynomial time," *Journal of Computer and System Sciences*, vol. 25, no. 1, pp. 42–65, 1982.

[104] M. Fleury and C. Weidenbach, "A verified SAT solver framework including optimization and partial valuations," in *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020* (E. Albert and L. Kovács, eds.), vol. 73 of *EPiC Series in Computing*, pp. 212–229, EasyChair, 2020.

[105] D. Oe, A. Stump, C. Oliver, and K. Clancy, "versat: A verified modern SAT solver," in *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings* (V. Kuncak and A. Rybalchenko, eds.), vol. 7148 of *Lecture Notes in Computer Science*, pp. 363–378, Springer, 2012.

[106] F. Maric, "Formal verification of a modern SAT solver by shallow embedding into isabelle/HOL," *Theoretical Computer Science*, vol. 411, no. 50, pp. 4333–4356, 2010.

[107] N. Wetzler, M. Heule, and W. Hunt, "DRAT-trim: Efficient checking and trimming using expressive clausal proofs," in *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings* (C. Sinz and U. Egly, eds.), vol. 8561 of *Lecture Notes in Computer Science*, pp. 422–429, Springer, 2014.

[108] N. Wetzler, M. Heule, and W. Hunt, “Mechanical verification of SAT refutations with extended resolution,” in *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings* (S. Blazy, C. Paulin-Mohring, and D. Pichardie, eds.), vol. 7998 of *Lecture Notes in Computer Science*, pp. 229–244, Springer, 2013.

[109] E. Goldberg and Y. Novikov, “Verification of proofs of unsatisfiability for CNF formulas,” in *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pp. 10886–10891, IEEE Computer Society, 2003.

[110] P. Lammich, “Efficient verified (UN)SAT certificate checking,” *Journal of Automated Reasoning*, vol. 64, no. 3, pp. 513–532, 2020.

[111] A. Darbari, B. Fischer, and J. Marques-Silva, “Industrial-strength certified SAT solving through verified SAT proof checking,” in *Theoretical Aspects of Computing - ICTAC 2010, 7th International Colloquium, Natal, Rio Grande do Norte, Brazil, September 1-3, 2010. Proceedings* (A. Cavalcanti, D. Déharbe, M. Gaudel, and J. Woodcock, eds.), vol. 6255 of *Lecture Notes in Computer Science*, pp. 260–274, Springer, 2010.

[112] E. Demaine and M. Demaine, “Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity,” *Graphs and Combinatorics*, vol. 23 (Supplement), pp. 195–208, 2007.

[113] M. Heule, “Solving edge-matching problems with satisfiability solvers,” in *Proceedings of the Second International Workshop on Logic and Search (LaSh 2008)*, pp. 88–102, University of Leuven, 2008.

[114] C. Ansótegui, R. Béjar, C. Fernández, and C. Mateu, “On the hardness of solving edge matching puzzles as SAT or CSP problems,” *Constraints*, vol. 18, no. 1, pp. 7–37, 2013.

[115] F. Jaeger, “A survey of the cycle double cover conjecture,” in *North-Holland Mathematics Studies*, vol. 115, pp. 1–12, Elsevier, 1985.

[116] J. Hägglund and K. Markström, “On stable cycles and cycle double covers of graphs with large circumference,” *Discrete Mathematicss*, vol. 312, no. 17, pp. 2540–2544, 2012.

[117] “Finnish Grid and Cloud Infrastructure,” 2019. urn:nbn:fi:research-infras-2016072533.

Appendix A

Group Representations

In this appendix we list for the readers' convenience the representations of the 23 groups constructed by Kangaslampi and Vdovina in [57], studied by the aforementioned authors in [56] as well as the present authors in this work. These groups are the only torsion-free groups acting simply transitively on the nodes of hyperbolic triangular buildings with the smallest generalized guadrangle as the link. The representations of the groups consist of generators $x_1, \ldots, x_{15}$ as well as the relations $x_i x_j x_k = 1$ listed below the name of each group as triplets (x_i, x_j, x_k).

T_1	T_2	T_3	T_4	T_5	T_6
(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)
(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)
(x_{11}, x_9, x_2)	(x_{11}, x_9, x_2)	(x_{11}, x_3, x_2)	(x_{11}, x_3, x_2)	(x_{11}, x_4, x_2)	(x_{11}, x_4, x_2)
(x_{14}, x_3, x_2)	(x_{14}, x_3, x_2)	(x_{14}, x_5, x_2)	(x_{14}, x_5, x_2)	(x_{14}, x_3, x_2)	(x_{14}, x_5, x_2)
(x_7, x_4, x_3)	(x_7, x_4, x_3)	(x_7, x_4, x_3)	(x_7, x_4, x_3)	(x_8, x_6, x_3)	(x_4, x_7, x_3)
(x_{15}, x_{13}, x_3)	(x_{15}, x_{13}, x_3)	(x_{15}, x_8, x_3)	(x_{15}, x_8, x_3)	(x_{14}, x_8, x_3)	(x_7, x_6, x_3)
(x_8, x_6, x_4)	(x_8, x_6, x_4)	(x_8, x_9, x_4)	(x_8, x_9, x_4)	(x_7, x_5, x_4)	(x_{12}, x_{12}, x_3)
(x_{12}, x_{11}, x_4)	(x_{12}, x_{11}, x_4)	(x_{12}, x_{12}, x_4)	(x_{12}, x_{13}, x_4)	(x_{15}, x_{13}, x_4)	(x_{15}, x_9, x_4)
(x_5, x_8, x_5)	(x_5, x_8, x_5)	(x_9, x_6, x_5)	(x_9, x_6, x_5)	(x_6, x_9, x_5)	(x_8, x_8, x_5)
(x_{10}, x_{12}, x_5)	(x_{10}, x_{12}, x_5)	(x_{13}, x_{13}, x_5)	(x_{13}, x_{12}, x_5)	(x_{14}, x_{12}, x_5)	(x_{14}, x_{13}, x_5)
(x_6, x_{14}, x_6)	(x_7, x_{14}, x_6)	(x_8, x_{11}, x_6)	(x_8, x_{11}, x_6)	(x_{11}, x_{12}, x_6)	(x_9, x_{14}, x_6)
(x_7, x_{12}, x_7)	(x_{12}, x_7, x_6)	(x_{10}, x_{13}, x_6)	(x_{10}, x_{12}, x_6)	(x_7, x_{11}, x_7)	(x_{11}, x_9, x_6)
(x_{13}, x_9, x_8)	(x_{13}, x_9, x_8)	(x_9, x_{14}, x_7)	(x_9, x_{14}, x_7)	(x_{15}, x_9, x_8)	(x_{15}, x_{13}, x_7)
(x_{14}, x_{15}, x_9)	(x_{14}, x_{15}, x_9)	(x_{10}, x_{12}, x_7)	(x_{10}, x_{13}, x_7)	(x_{10}, x_{13}, x_9)	(x_{10}, x_{12}, x_8)
(x_{13}, x_{11}, x_{10})	(x_{13}, x_{11}, x_{10})	(x_{15}, x_{14}, x_{11})	(x_{15}, x_{14}, x_{11})	(x_{12}, x_{13}, x_{10})	(x_{13}, x_{11}, x_{10})

T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)	(x_1, x_{10}, x_1)
(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)	(x_{15}, x_2, x_1)
(x_{11}, x_5, x_2)	(x_{11}, x_4, x_2)	(x_{11}, x_4, x_2)	(x_{11}, x_8, x_2)	(x_{11}, x_6, x_2)	(x_{11}, x_3, x_2)
(x_{14}, x_4, x_2)	(x_{14}, x_7, x_2)	(x_{14}, x_6, x_2)	(x_{14}, x_5, x_2)	(x_{14}, x_4, x_2)	(x_{14}, x_9, x_2)
(x_4, x_7, x_3)	(x_5, x_{12}, x_3)	(x_5, x_9, x_3)	(x_3, x_{11}, x_3)	(x_5, x_7, x_3)	(x_9, x_{14}, x_3)
(x_7, x_6, x_3)	(x_8, x_5, x_3)	(x_8, x_7, x_3)	(x_9, x_7, x_3)	(x_8, x_{12}, x_3)	(x_{13}, x_7, x_3)
(x_{12}, x_{12}, x_3)	(x_{10}, x_{13}, x_3)	(x_{10}, x_{13}, x_3)	(x_5, x_9, x_4)	(x_{10}, x_{13}, x_3)	(x_4, x_{12}, x_4)
(x_{15}, x_{13}, x_4)	(x_7, x_9, x_4)	(x_8, x_5, x_4)	(x_7, x_6, x_4)	(x_9, x_9, x_4)	(x_7, x_6, x_4)
(x_8, x_8, x_5)	(x_{15}, x_{14}, x_4)	(x_{14}, x_{14}, x_4)	(x_{11}, x_{12}, x_4)	(x_{13}, x_8, x_4)	(x_5, x_8, x_5)
(x_{14}, x_9, x_5)	(x_8, x_6, x_5)	(x_{10}, x_{12}, x_5)	(x_{13}, x_{13}, x_5)	(x_5, x_{11}, x_5)	(x_{10}, x_{13}, x_5)
(x_9, x_{11}, x_6)	(x_7, x_{13}, x_6)	(x_7, x_{12}, x_6)	(x_9, x_{12}, x_6)	(x_8, x_7, x_6)	(x_9, x_8, x_6)
(x_{11}, x_{13}, x_6)	(x_{11}, x_9, x_6)	(x_{15}, x_9, x_6)	(x_{10}, x_{13}, x_6)	(x_{14}, x_{14}, x_6)	(x_{10}, x_{12}, x_6)
(x_{15}, x_9, x_7)	(x_{13}, x_{15}, x_8)	(x_8, x_{11}, x_7)	(x_{15}, x_8, x_7)	(x_{15}, x_{15}, x_7)	(x_{15}, x_{15}, x_7)
(x_{10}, x_{12}, x_8)	(x_{14}, x_{12}, x_9)	(x_{15}, x_{13}, x_9)	(x_{15}, x_{14}, x_8)	(x_{10}, x_{12}, x_9)	(x_{13}, x_{11}, x_8)
(x_{13}, x_{14}, x_{10})	(x_{12}, x_{11}, x_{10})	(x_{12}, x_{13}, x_{11})	(x_{12}, x_{14}, x_{10})	(x_{13}, x_{12}, x_{11})	(x_{12}, x_{14}, x_{11})

T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}
(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)	(x_1, x_{15}, x_1)
(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)	(x_{10}, x_2, x_1)
(x_{11}, x_3, x_2)	(x_{11}, x_3, x_2)	(x_{11}, x_5, x_2)	(x_{11}, x_5, x_2)	(x_{11}, x_4, x_2)	(x_{11}, x_4, x_2)
(x_{14}, x_4, x_2)	(x_{14}, x_5, x_2)	(x_{14}, x_4, x_2)	(x_{14}, x_3, x_2)	(x_{14}, x_6, x_2)	(x_{14}, x_3, x_2)
(x_7, x_5, x_3)	(x_7, x_4, x_3)	(x_3, x_6, x_3)	(x_8, x_4, x_3)	(x_3, x_{12}, x_3)	(x_9, x_5, x_3)
(x_{15}, x_{12}, x_3)	(x_{15}, x_{12}, x_3)	(x_{15}, x_{12}, x_3)	(x_{14}, x_9, x_3)	(x_8, x_5, x_3)	(x_{13}, x_7, x_3)
(x_8, x_{13}, x_4)	(x_8, x_6, x_4)	(x_7, x_8, x_4)	(x_6, x_6, x_4)	(x_8, x_{13}, x_4)	(x_8, x_6, x_4)
(x_{14}, x_9, x_4)	(x_{12}, x_9, x_4)	(x_{15}, x_{13}, x_4)	(x_{15}, x_{13}, x_4)	(x_{14}, x_{14}, x_4)	(x_{14}, x_8, x_4)
(x_9, x_6, x_5)	(x_8, x_{13}, x_5)	(x_8, x_7, x_5)	(x_7, x_7, x_5)	(x_9, x_7, x_5)	(x_6, x_{12}, x_5)
(x_{11}, x_8, x_5)	(x_{14}, x_8, x_5)	(x_{14}, x_9, x_5)	(x_{15}, x_{12}, x_5)	(x_{11}, x_9, x_5)	(x_{15}, x_{13}, x_5)
(x_7, x_8, x_6)	(x_7, x_9, x_6)	(x_9, x_{11}, x_6)	(x_{14}, x_{11}, x_6)	(x_7, x_8, x_6)	(x_7, x_9, x_6)
(x_{11}, x_{12}, x_6)	(x_{12}, x_{11}, x_6)	(x_{11}, x_{13}, x_6)	(x_{11}, x_{13}, x_7)	(x_{15}, x_{12}, x_6)	(x_{11}, x_{10}, x_7)
(x_{10}, x_{13}, x_7)	(x_{10}, x_{13}, x_7)	(x_{10}, x_9, x_7)	(x_9, x_{12}, x_8)	(x_{10}, x_{13}, x_7)	(x_{14}, x_{12}, x_8)
(x_{14}, x_{10}, x_9)	(x_{14}, x_{10}, x_9)	(x_{12}, x_{12}, x_8)	(x_{10}, x_9, x_8)	(x_{11}, x_{10}, x_9)	(x_{13}, x_{11}, x_9)
(x_{15}, x_{13}, x_{12})	(x_{15}, x_{13}, x_{11})	(x_{13}, x_{14}, x_{10})	(x_{13}, x_{12}, x_{10})	(x_{15}, x_{13}, x_{12})	(x_{15}, x_{12}, x_{10})

T_{19}	T_{20}	T_{21}	T_{22}	T_{23}
(x_1, x_{15}, x_1)	(x_1, x_{10}, x_1)	(x_5, x_2, x_1)	(x_4, x_2, x_1)	(x_4, x_2, x_1)
(x_{10}, x_2, x_1)	(x_{15}, x_6, x_1)	(x_6, x_4, x_1)	(x_7, x_3, x_1)	(x_6, x_5, x_1)
(x_{11}, x_6, x_2)	(x_3, x_7, x_2)	(x_{13}, x_3, x_1)	(x_{12}, x_5, x_1)	(x_{14}, x_3, x_1)
(x_{14}, x_9, x_2)	(x_8, x_9, x_2)	(x_{10}, x_7, x_2)	(x_{10}, x_{13}, x_2)	(x_{10}, x_7, x_2)
(x_4, x_{11}, x_3)	(x_{12}, x_8, x_2)	(x_{15}, x_{12}, x_2)	(x_{15}, x_{10}, x_2)	(x_{15}, x_{11}, x_2)
(x_7, x_4, x_3)	(x_5, x_4, x_3)	(x_{11}, x_{14}, x_3)	(x_{11}, x_6, x_3)	(x_{11}, x_8, x_3)
(x_{12}, x_5, x_3)	(x_{11}, x_{14}, x_3)	(x_{14}, x_8, x_3)	(x_{14}, x_8, x_3)	(x_{14}, x_{12}, x_3)
(x_9, x_{14}, x_4)	(x_6, x_{11}, x_4)	(x_{12}, x_{15}, x_4)	(x_7, x_{15}, x_4)	(x_9, x_{13}, x_4)
(x_8, x_{13}, x_5)	(x_{15}, x_{13}, x_4)	(x_{13}, x_{11}, x_4)	(x_{15}, x_9, x_4)	(x_{13}, x_{10}, x_4)
(x_{12}, x_8, x_5)	(x_9, x_{15}, x_5)	(x_9, x_{10}, x_5)	(x_{12}, x_{11}, x_5)	(x_{12}, x_{15}, x_5)
(x_9, x_8, x_6)	(x_{10}, x_{12}, x_5)	(x_{13}, x_9, x_5)	(x_{13}, x_{14}, x_5)	(x_{13}, x_9, x_5)
(x_{13}, x_7, x_6)	(x_{14}, x_{11}, x_6)	(x_9, x_8, x_6)	(x_8, x_9, x_6)	(x_8, x_9, x_6)
(x_{14}, x_{10}, x_7)	(x_8, x_{13}, x_7)	(x_{10}, x_{11}, x_6)	(x_{12}, x_7, x_6)	(x_{10}, x_{12}, x_6)
(x_{15}, x_{12}, x_{10})	(x_{14}, x_9, x_7)	(x_8, x_{15}, x_7)	(x_{11}, x_{13}, x_8)	(x_7, x_{15}, x_7)
(x_{15}, x_{13}, x_{11})	(x_{13}, x_{12}, x_{10})	(x_{14}, x_{12}, x_7)	(x_{14}, x_{10}, x_9)	(x_{11}, x_{14}, x_8)

Appendix B

Witnesses

In Figures B.1, B.2, and B.3 we show example graphs from $\mathsf{labels}(T_6, 3)$, $\mathsf{labels}(T_{13}, 3)$, and $\mathsf{labels}(T_{16}, 3)$, respectively. In Figure B.4 we present an example of a graph in $\mathsf{labels}(T_{15}, 4)$. The arrows around the nodes of the graphs denote the orientations arising from the cycleset allowing the graph to be labeled. Bipartiteness is indicated using colors black and white, and the x_i's denote the labels of edges. The triplet of each node can be deduced from the labels of its incident edges.

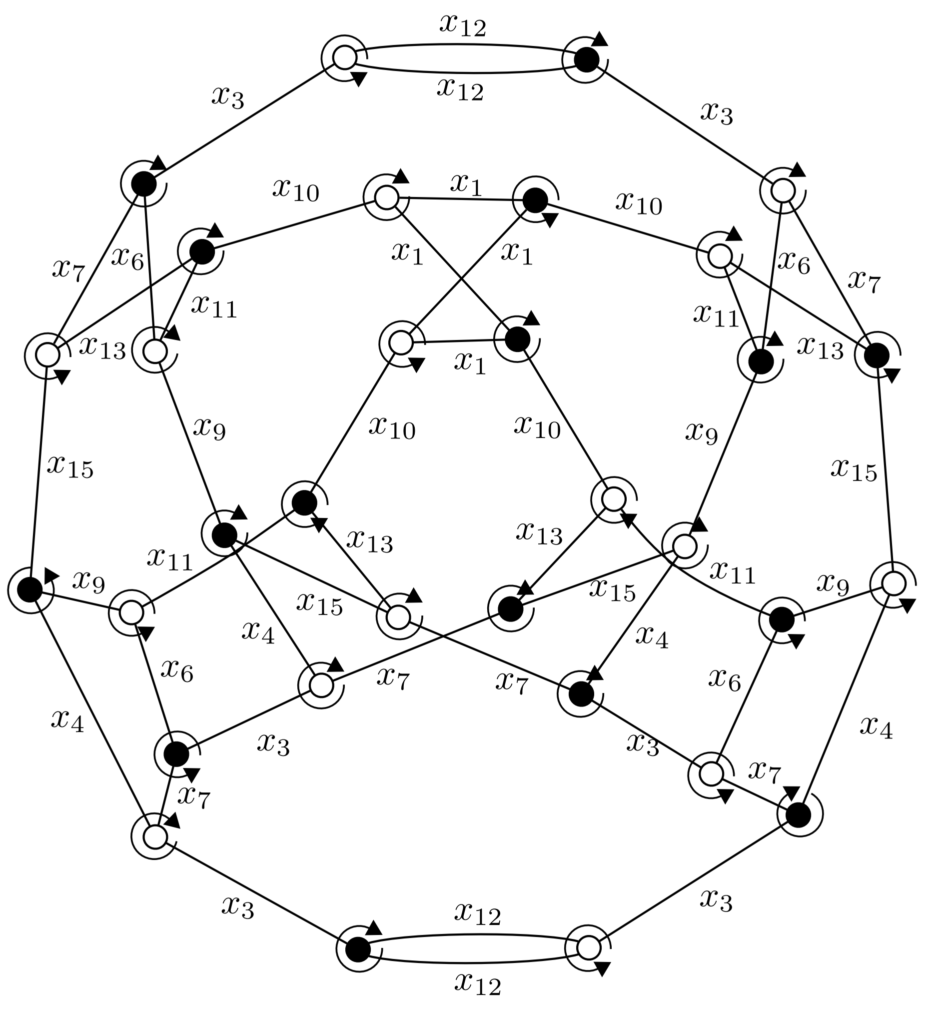


Figure B.1: Graph G^3_{2668} labeled using T_6.

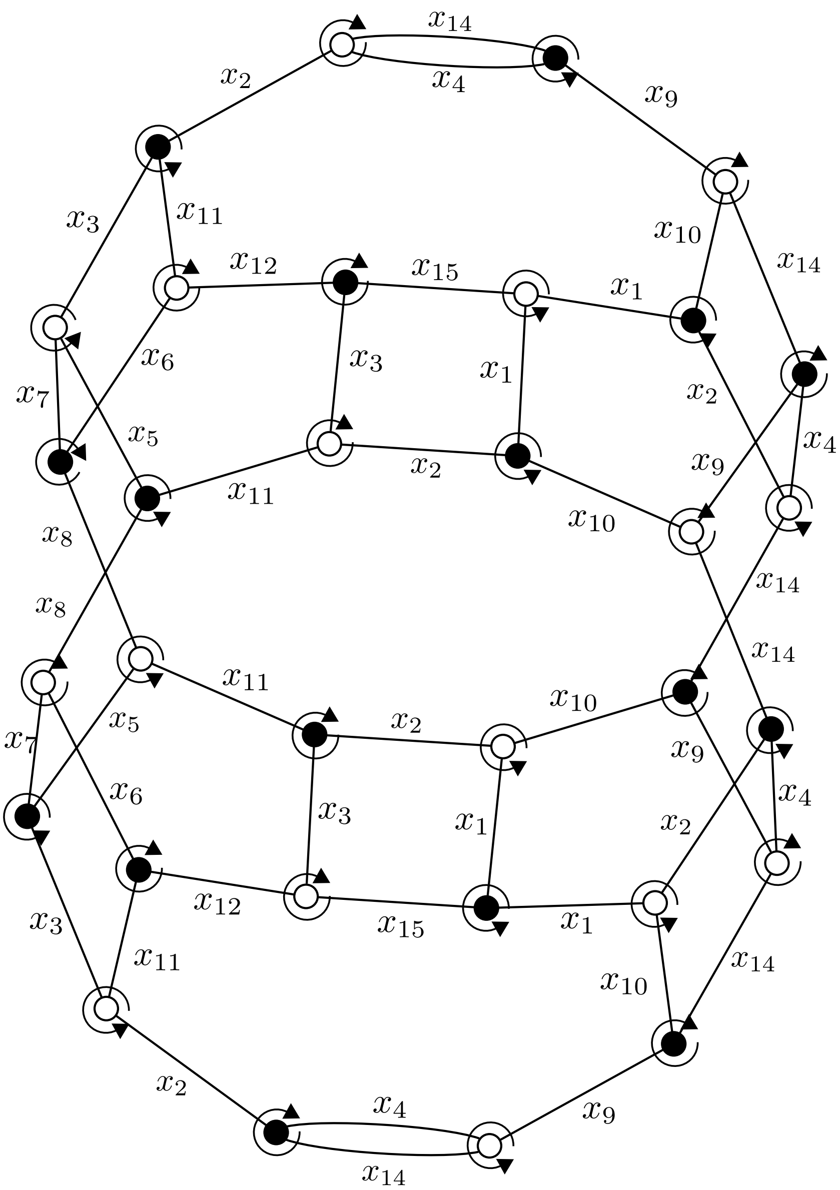


Figure B.2: Graph G^3_{2056} labeled using T_{13}.

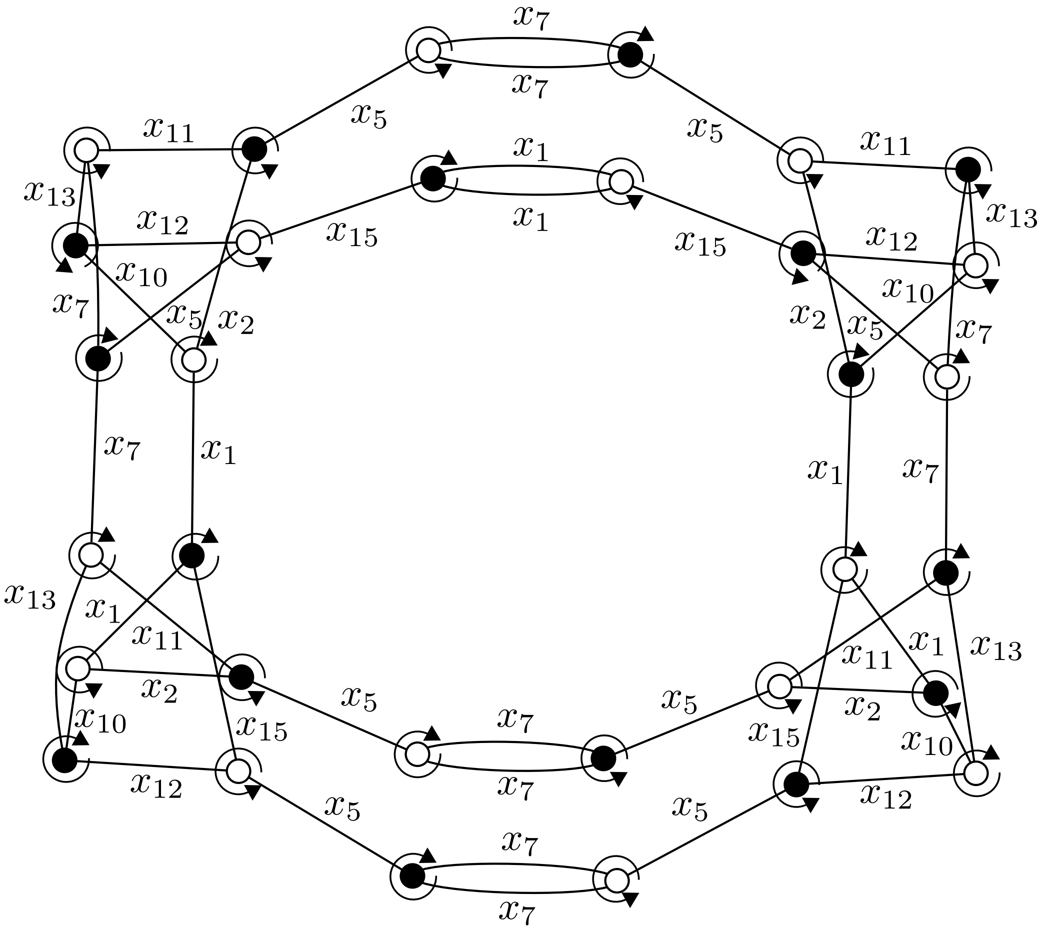


Figure B.3: Graph G^3_{2211} labeled using T_{16}.

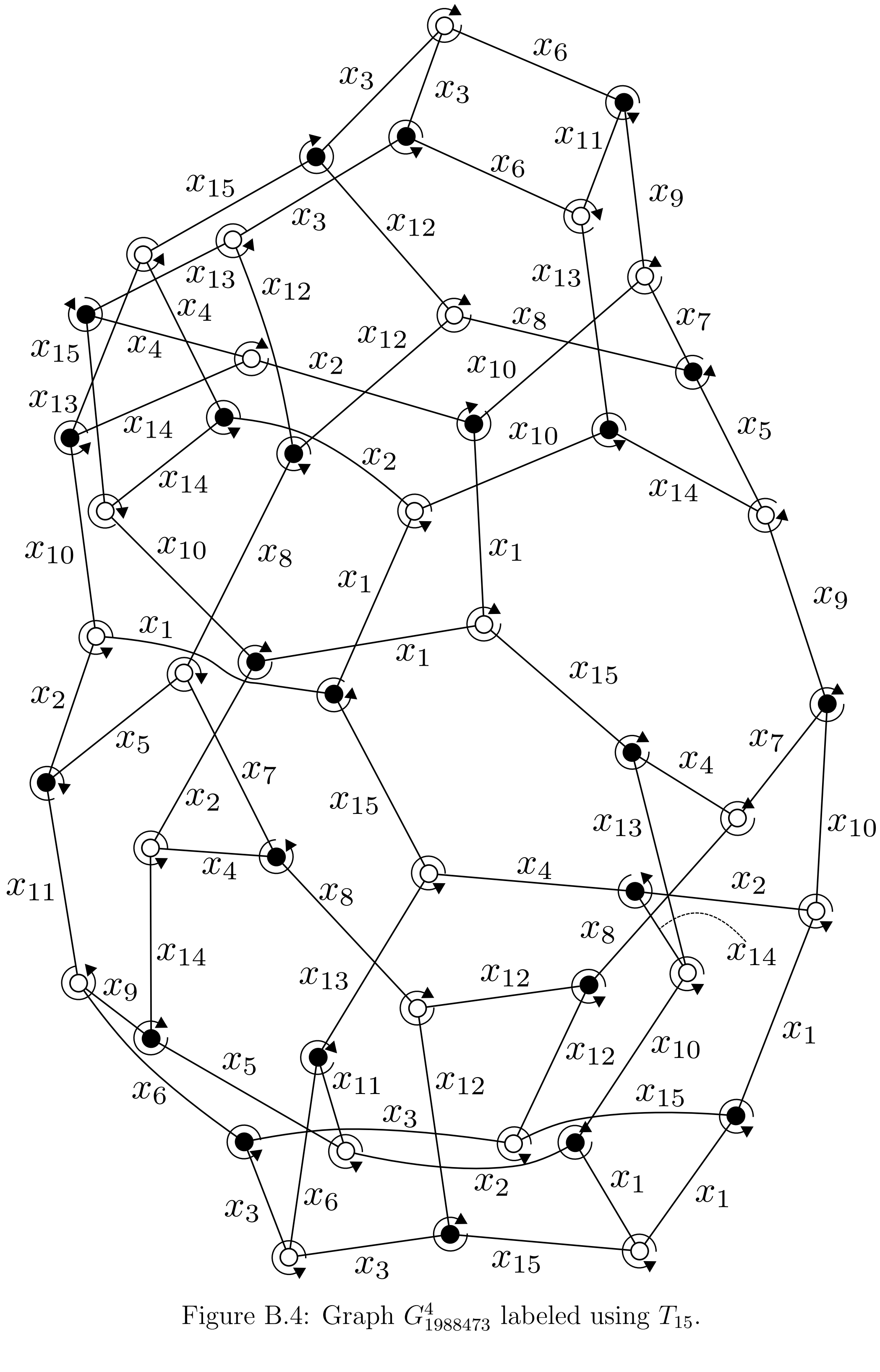


Figure B.4: Graph $G^4_{1988473}$ labeled using T_{15}.