Generating Optimal 1-Planar Graphs for Automated Conjecture-Making

David Scholz

January 15, 2021

TH Köln

Contents

Introduction

Introduction

All considered graphs are finite, simple and undirected.

Introduction

All considered graphs are finite, simple and undirected.

- A conjecture is a formal statement which is initially open to formal proofs,

Introduction

All considered graphs are finite, simple and undirected.

- A conjecture is a formal statement which is initially open to formal proofs,
- Conjectures in form of inequalities: $I \leq J$, $I \leq J + K$, $I + J \leq K + L$ with I, J, K, L being invariants ($\chi \leq \Delta + 1$ (Brook's theorem)),

Introduction

All considered graphs are finite, simple and undirected.

- A conjecture is a formal statement which is initially open to formal proofs,
- Conjectures in form of inequalities: $I \leq J$, $I \leq J + K$, $I + J \leq K + L$ with I, J, K, L being invariants ($\chi \leq \Delta + 1$ (Brook's theorem)),
- Graph generation is the process of recursively applying some kind of rules in order to expand a smaller graph to a larger one without leaving the underlying class.

Optimal 1-planar graphs

Optimal 1-planar graphs

Definition 1
A graph is 1-planar if it can be drawn in the plane so that each edge is crossed by at most one other edge.

Optimal 1-planar graphs

Definition 1
A graph is 1-planar if it can be drawn in the plane so that each edge is crossed by at most one other edge.

Lemma 1
A planar graph G on $n \geq 3$ vertices has at most $3n - 6$ edges.

Definition 1
A graph is 1-planar if it can be drawn in the plane so that each edge is crossed by at most one other edge.

Lemma 1
A planar graph G on $n \geq 3$ vertices has at most $3n - 6$ edges.

These are the plane graphs for which each face is a triangle (plane triangulations).

Optimal 1-planar graphs

Theorem 1 (Ringel)
Let G be a 1-planar graph on n vertices and m edges. Then $m \leq 4n - 8$.

Theorem 1 (Ringel)
Let G be a 1-planar graph on n vertices and m edges. Then $m \leq 4n - 8$.

Proof.
Assume that G is maximal and $\widetilde{G}$ is an embedding of G. Denote c as the number of crossings in $\widetilde{G}$.

Theorem 1 (Ringel)
Let G be a 1-planar graph on n vertices and m edges. Then $m \leq 4n - 8$.

Proof.
Assume that G is maximal and $\widetilde{G}$ is an embedding of G. Denote c as the number of crossings in $\widetilde{G}$.

Now, if (v_1, v_2) and (v_3, v_4) in $E(\widetilde{G})$ cross, then consequently $\{(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$ are in $E(G)$ due to maximality of G.

Optimal 1-planar graphs

Theorem 1 (Ringel)
Let G be a 1-planar graph on n vertices and m edges. Then $m \leq 4n - 8$.

Proof.
Assume that G is maximal and $\widetilde{G}$ is an embedding of G. Denote c as the number of crossings in $\widetilde{G}$.

Now, if (v_1, v_2) and (v_3, v_4) in $E(\widetilde{G})$ cross, then consequently $\{(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$ are in $E(G)$ due to maximality of G.

Hence, if we delete an edge from each pair of crossing edges, the resulting graph H is a plane triangulation on $n' = n$ vertices, m' edges and f' faces.

Optimal 1-planar graphs

By lemma 1 we have $m' \leq 3n' - 6$, therefore

$$n' - m' + f' \leq 2 \tag{1}$$

$$n' - 3n' + 6 + f' \leq 2 \tag{2}$$

$$f' \leq 2n' - 4 \tag{3}$$

□

Optimal 1-planar graphs

By lemma 1 we have $m' \leq 3n' - 6$, therefore

$$
\begin{aligned}
n' - m' + f' &\leq 2 && (1)\\
n' - 3n' + 6 + f' &\leq 2 && (2)\\
f' &\leq 2n' - 4 && (3)
\end{aligned}
$$

Now, $m = m' + c \leq m' + \frac{f'}{2} \leq m' + \frac{2n'-4}{2} = m' + n' - 2 \leq 3n' - 6 + n' - 2 = 4n - 8$ $\square$

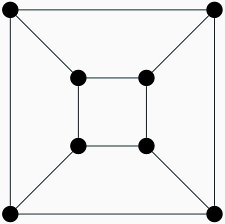
Optimal 1-planar graphs

By proof construction we realize that each pair of crossings form diagonal edges in a 4-cycle.

Optimal 1-planar graphs

By proof construction we realize that each pair of crossings form diagonal edges in a 4-cycle.

Definition 2
A (simple) quadrangulation of the sphere is a graph embedded on the sphere, such that every face is bounded by a 4-cycle.



Lemma 2
Given a simple quadrangulation $Q = (V, E)$, then for the average degree holds $\frac{1}{|V|} \sum_{v \in V} d(v) < 4$.

Optimal 1-planar graphs

Lemma 2
Given a simple quadrangulation $Q = (V, E)$, then for the average degree holds $\frac{1}{|V|}\sum_{v \in V} d(v) < 4$.

Proof.
We denote m as the number of edges, n as the number of vertices and f as the number of faces and assume $\frac{1}{n}\sum_{v \in V} d(v) \geq 4$.

Lemma 2
Given a simple quadrangulation $Q = (V, E)$, then for the average degree holds $\frac{1}{|V|}\sum_{v \in V} d(v) < 4$.

Proof.
We denote m as the number of edges, n as the number of vertices and f as the number of faces and assume $\frac{1}{n}\sum_{v \in V} d(v) \geq 4$.

$$\iff \sum_{v \in V} d(v) \geq 4n \tag{4}$$

$$\iff 2m \geq 4n \quad \text{(handshaking lemma)} \tag{5}$$

$$\iff \frac{1}{2}m \geq n \tag{6}$$

Optimal 1-planar graphs

Using Euler's formula ($n - m + f = 2$) yields to

$$
\begin{aligned}
\frac{1}{2}m - m + f &\geq 2 & (7)\\
\iff f - \frac{1}{2}m &\geq 2 & (8)\\
\iff f &\geq 2 + \frac{m}{2} & (9)
\end{aligned}
$$

$\square$

Optimal 1-planar graphs

Using Euler's formula ($n - m + f = 2$) yields to

$$\begin{aligned}
\frac{1}{2}m - m + f &\geq 2 && (7)\\
\iff f - \frac{1}{2}m &\geq 2 && (8)\\
\iff f &\geq 2 + \frac{m}{2} && (9)
\end{aligned}$$

Since every quadrangulation is bounded by a 4-cycle, each edge is used in 2 faces. Hence $m = \frac{4}{2}f \iff f = \frac{1}{2}m$.

□

Optimal 1-planar graphs

Using Euler's formula ($n - m + f = 2$) yields to

$$\begin{aligned} \frac{1}{2}m - m + f &\geq 2 && (7)\\ \iff f - \frac{1}{2}m &\geq 2 && (8)\\ \iff f &\geq 2 + \frac{m}{2} && (9) \end{aligned}$$

Since every quadrangulation is bounded by a 4-cycle, each edge is used in 2 faces. Hence $m = \frac{4}{2}f \iff f = \frac{1}{2}m$.

Therefore, $\frac{1}{2}m \geq 2 + \frac{m}{2}$. $\square$

Using Euler's formula ($n - m + f = 2$) yields to

$$
\begin{aligned}
\frac{1}{2}m - m + f &\geq 2 && (7)\\
\iff f - \frac{1}{2}m &\geq 2 && (8)\\
\iff f &\geq 2 + \frac{m}{2} && (9)
\end{aligned}
$$

Since every quadrangulation is bounded by a 4-cycle, each edge is used in 2 faces. Hence $m = \frac{4}{2}f \iff f = \frac{1}{2}m$.

Therefore, $\frac{1}{2}m \geq 2 + \frac{m}{2}$. A contradiction. $\square$

Optimal 1-planar graphs

Corollary 1
The connectivity of a simple quadrangulation cannot be more than 3.

Corollary 1
The connectivity of a simple quadrangulation cannot be more than 3.

Proof.
Given a simple quadrangulation $Q = (V, E)$ with $|V| = n$. Assume that for each $v \in V$, $d(v) = 4$. Then

$$\frac{1}{n}\sum_{v \in V} d(v) = \frac{4n}{n} = 4$$

Corollary 1
The connectivity of a simple quadrangulation cannot be more than 3.

Proof.
Given a simple quadrangulation $Q = (V, E)$ with $|V| = n$. Assume that for each $v \in V$, $d(v) = 4$. Then

$$\frac{1}{n}\sum_{v \in V} d(v) = \frac{4n}{n} = 4$$

A contradiction to lemma 2. Hence, there must be a vertex $v' \in V$ with $d(v') \leq 3$. Its neighborhood form a cutset. $\square$

Optimal 1-planar graphs

We call the quadrangulations obtained by removing the crossing edges from an optimal 1-planar graph its skeleton.

Optimal 1-planar graphs

We call the quadrangulations obtained by removing the crossing edges from an optimal 1-planar graph its skeleton.

- It has been further proven by Bodendiek et. al and later by Suzuki that the skeletons of the optimal 1-planar graphs are 3-connected,

Optimal 1-planar graphs

We call the quadrangulations obtained by removing the crossing edges from an optimal 1-planar graph its skeleton.

- It has been further proven by Bodendiek et. al and later by Suzuki that the skeletons of the optimal 1-planar graphs are 3-connected,
- The number of vertices in an optimal 1-planar graph is at least 8 (this is easily proven by using $4n - 8 \leq \frac{n(n-1)}{2}$),

Optimal 1-planar graphs

We call the quadrangulations obtained by removing the crossing edges from an optimal 1-planar graph its skeleton.

- It has been further proven by Bodendiek et. al and later by Suzuki that the skeletons of the optimal 1-planar graphs are 3-connected,
- The number of vertices in an optimal 1-planar graph is at least 8 (this is easily proven by using $4n - 8 \leq \frac{n(n-1)}{2}$),
- There are no optimal 1-planar graphs with 9 vertices.

Generating optimal 1-planar graphs

Generating optimal 1-planar graphs

Idea: Generate the simple 3-connected quadrangulations and add the crossing diagonals in each face.

Generating optimal 1-planar graphs

Idea: Generate the simple 3-connected quadrangulations and add the crossing diagonals in each face.

Definition 3
Given a class $\mathcal{C}$ of graphs and a set of expansions $\mathcal{P} := \{P_0, P_1, \cdots, P_k\}$. The relation $R(\mathcal{C}, \mathcal{P})$ is defined by

$$R(\mathcal{C}, \mathcal{P}) := \{(G, G') \in \mathcal{C} \times \mathcal{C} \mid G' \text{ can be obtained from G by applying some } P \in \mathcal{P}\}$$

Generating optimal 1-planar graphs

We define an expansion as its inverse, called reduction.

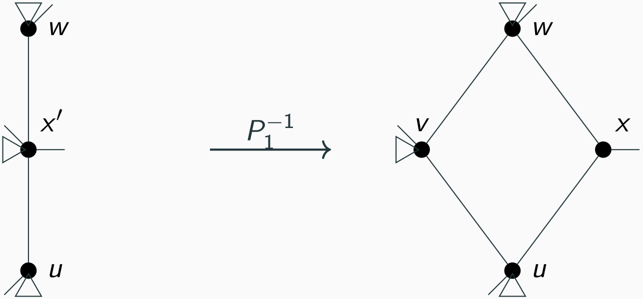
We define an expansion as its inverse, called reduction.

Definition 4
A P_1-reduction consists of a contraction of a face (x, u, v, w) at $\{x, v\}$ whereby x has degree 3 and u, v, w each have degree at least 3.

Generating optimal 1-planar graphs

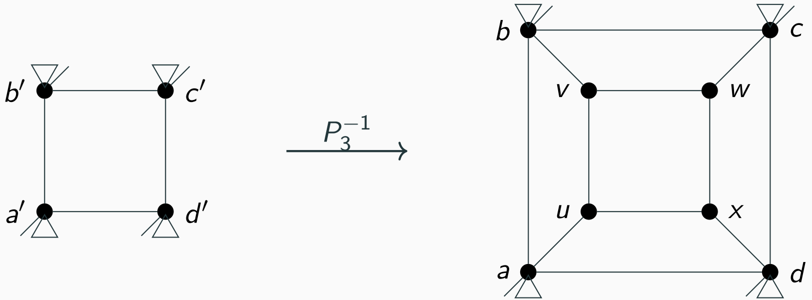
We define an expansion as its inverse, called reduction.

Definition 4
A P_1-reduction consists of a contraction of a face (x, u, v, w) at $\{x, v\}$ whereby x has degree 3 and u, v, w each have degree at least 3.



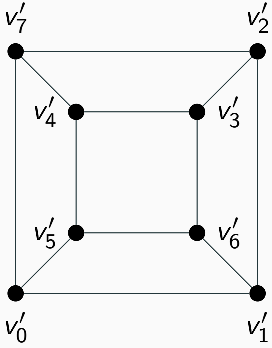
Definition 5
A P_3-reduction is a sequence of four contractions. We have faces $(u, v, w, x), (a, b, v, u), (b, c, w, v), (c, d, x, w)$ and (d, a, u, x). The vertices u, v, w, x all have degree 3, and we assume that a, b, c, d all have degree of at least 4. A P_3-reduction consists of a face contraction at $\{a, v\}$, followed by one at $\{b, w\}$, followed by one at $\{c, x\}$, followed by one at $\{d, u\}$.

Definition 5
A P_3-reduction is a sequence of four contractions. We have faces $(u, v, w, x), (a, b, v, u), (b, c, w, v), (c, d, x, w)$ and (d, a, u, x). The vertices u, v, w, x all have degree 3, and we assume that a, b, c, d all have degree of at least 4. A P_3-reduction consists of a face contraction at $\{a, v\}$, followed by one at $\{b, w\}$, followed by one at $\{c, x\}$, followed by one at $\{d, u\}$.

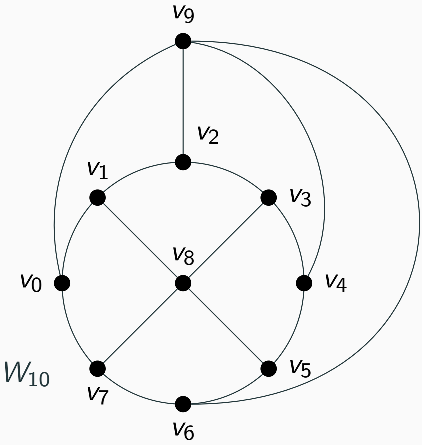


Definition 6
Given a simple quadrangulation Q with $n \geq 8$ vertices, whereby n is even. Q consists of a cycle $v_0 v_1 \cdots v_{n-3}$, as well as a vertex which is adjacent to $v_0, v_2, \cdots, v_{n-4}$ and a vertex which is adjacent to $v_1, v_3, \cdots, v_{n-3}$. We call Q a pseudo-double wheel. The smallest one is a cube.

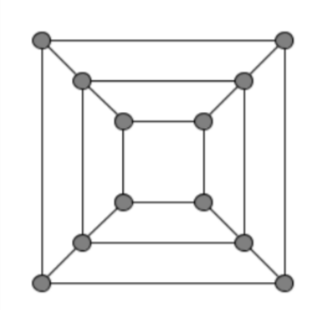
Definition 6
Given a simple quadrangulation Q with $n \geq 8$ vertices, whereby n is even. Q consists of a cycle $v_0 v_1 \cdots v_{n-3}$, as well as a vertex which is adjacent to $v_0, v_2, \cdots, v_{n-4}$ and a vertex which is adjacent to $v_1, v_3, \cdots, v_{n-3}$. We call Q a pseudo-double wheel. The smallest one is a cube.



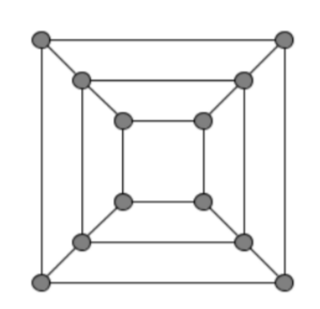
Generating optimal 1-planar graphs

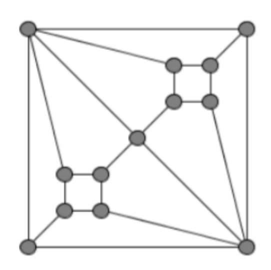


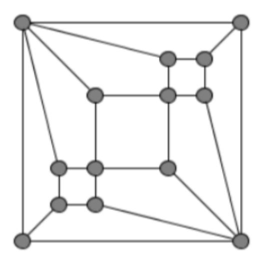
Generating optimal 1-planar graphs



Generating optimal 1-planar graphs







Generating optimal 1-planar graphs

Theorem 2
The class of all 3-connected quadrangulations is generated from the pseudo-double wheels by the P_1- and the P_3-expansions.

Generating optimal 1-planar graphs

Idea: Generate the simple quadrangulations and add crossing diagonals in each face.

Generating optimal 1-planar graphs

Idea: Generate the simple quadrangulations and add crossing diagonals in each face.

- How do we efficiently find all faces?

Idea: Generate the simple quadrangulations and add crossing diagonals in each face.

- How do we efficiently find all faces?
- How do we implement the generation algorithm?

Generating optimal 1-planar graphs

Idea: Generate the simple quadrangulations and add crossing diagonals in each face.

- How do we efficiently find all faces?
- How do we implement the generation algorithm?
- How do we store the generated quadrangulations?

Let $p : E \to V^2$ define two additional mappings $s, t : E \to V$ by $(s(e), t(e)) := p(e), \forall e \in E$. We call p the incidence mapping of a graph G.

Definition 7
For each $e \in E$ we define two directed edges $e^+ := (s(e), t(e))$ and $e^- := (t(e), s(e))$ and call them darts of e. We denote the sets of all darts of a graph G by D_E and call it dart relation on G.

Definition 8
A rotation for a vertex $v \in V$ is a single cyclic permutation $\pi_k = (e_1^+ e_2^+ \cdots e_k^+)$ of darts with $k \in \{1, \cdots, deg(v)\}$ and v being the source of each e_k^+. We denote the rotation of v by $rot(v)$.

Definition 8
A rotation for a vertex $v \in V$ is a single cyclic permutation $\pi_k = (e_1^+ e_2^+ \cdots e_k^+)$ of darts with $k \in \{1, \cdots, deg(v)\}$ and v being the source of each e_k^+. We denote the rotation of v by $rot(v)$.

Definition 9
A rotation system of G is a pair of permutations (α, σ) on D_E such that $\alpha = \Pi_{e \in E}(e^+ e^-)$ and $\sigma = \Pi_{v \in V} rot(v)$.

Definition 8
A rotation for a vertex $v \in V$ is a single cyclic permutation $\pi_k = (e_1^+ e_2^+ \cdots e_k^+)$ of darts with $k \in \{1, \cdots, deg(v)\}$ and v being the source of each e_k^+. We denote the rotation of v by $rot(v)$.

Definition 9
A rotation system of G is a pair of permutations (α, σ) on D_E such that $\alpha = \Pi_{e \in E}(e^+ e^-)$ and $\sigma = \Pi_{v \in V} rot(v)$.

Definition 10
Let (α, σ) be a rotation system for G and $\Phi = \alpha^{-1}\sigma^{-1}$. Then the cycles of Φ are called faces of G for this rotation system.

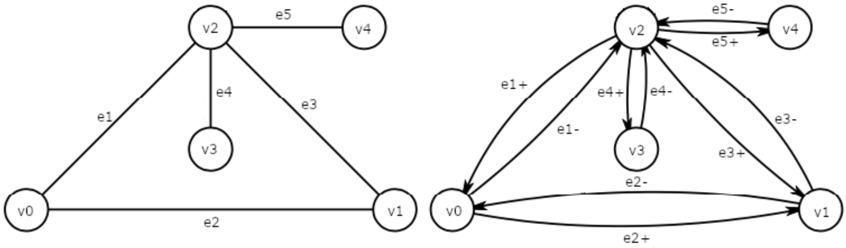
Conventions

- We compose permuations from left to rigth,

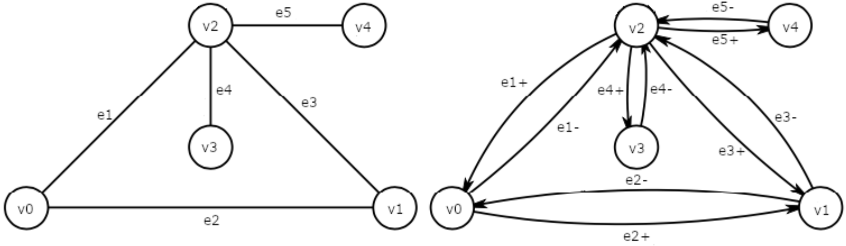
Conventions

- We compose permuations from left to rigth,
- Our rotations are given in cyclic clockwise order around its vertices.

Generating optimal 1-planar graphs

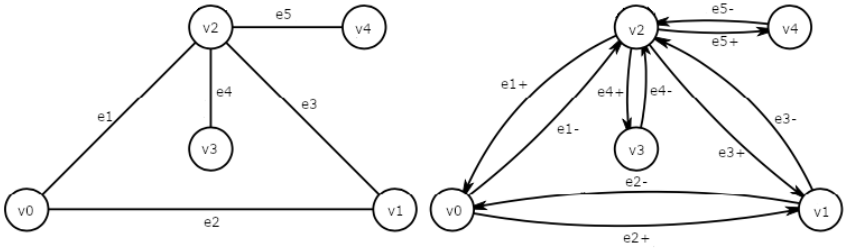


Generating optimal 1-planar graphs



$$\alpha = \Pi_{e \in E} = (e_1^+ e_1^-)(e_2^+ e_2^-)(e_3^+ e_3^-)(e_4^+ e_4^-)(e_5^+ e_5^-)$$

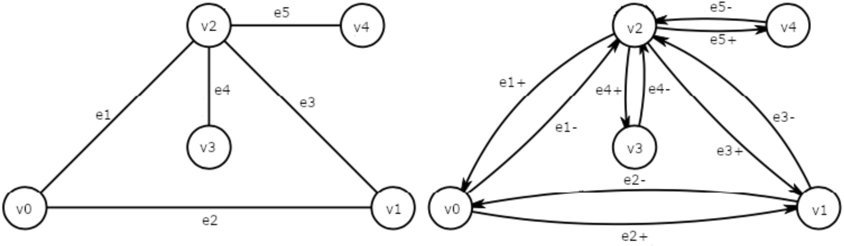
Generating optimal 1-planar graphs



$$\alpha = \Pi_{e \in E} = (e_1^+ e_1^-)(e_2^+ e_2^-)(e_3^+ e_3^-)(e_4^+ e_4^-)(e_5^+ e_5^-)$$

$$\sigma = \Pi_{v \in V} rot(v) = (e_5^+ e_3^+ e_4^+ e_1^+)(e_5^-)(e_3^- e_2^-)(e_4^-)(e_1^- e_2^+)$$

Generating optimal 1-planar graphs



$$\alpha = \Pi_{e \in E} = (e_1^+ e_1^-)(e_2^+ e_2^-)(e_3^+ e_3^-)(e_4^+ e_4^-)(e_5^+ e_5^-)$$

$$\sigma = \Pi_{v \in V} rot(v) = (e_5^+ e_3^+ e_4^+ e_1^+)(e_5^-)(e_3^- e_2^-)(e_4^-)(e_1^- e_2^+)$$
$$\Phi = \alpha^{-1}\sigma^{-1} = (e_3^+ e_2^- e_1^- e_4^+ e_4^-)(e_1^+ e_2^+ e_3^- e_5^+ e_5^-)$$

Generating optimal 1-planar graphs

- How do we store a rotation system?

Generating optimal 1-planar graphs

- How do we store a rotation system?
- We list neighbours in clockwise order

$1 : 3, 4$	$2 : 3, 5$
$3 : 1, 4, 5, 2$	$4 : 1, 5, 3$
$5 : 2, 3, 4$	

Generating optimal 1-planar graphs

- How do we store a rotation system?
- We list neighbours in clockwise order

$$1 : 3, 4 \qquad 2 : 3, 5$$
$$3 : 1, 4, 5, 2 \qquad 4 : 1, 5, 3$$
$$5 : 2, 3, 4$$

$\alpha =$
$((1,3)(3,1))((1,4)(4,1))((2,3)(3,2))((2,5)(5,2))((3,4)(4,3))$
$((3,5)(5,3))((4,5)(5,4))$

Generating optimal 1-planar graphs

- How do we store a rotation system?
- We list neighbours in clockwise order

$$1 : 3, 4 \qquad 2 : 3, 5$$
$$3 : 1, 4, 5, 2 \qquad 4 : 1, 5, 3$$
$$5 : 2, 3, 4$$

$\alpha =$
$((1,3)(3,1))((1,4)(4,1))((2,3)(3,2))((2,5)(5,2))((3,4)(4,3))$
$((3,5)(5,3))((4,5)(5,4))$
$\sigma =$
$((1,3)(1,4))((2,3)(2,5))((3,1)(3,4)(3,5)(3,2))((4,1)(4,5)(4,3))$
$((5,2)(5,3)(5,4))$

Generating optimal 1-planar graphs

Planar Code

Planar Code

- Start with a single byte equal to the number of vertices (let us denote it by n)
- We have n sections (one for each vertex)

Planar Code

- Start with a single byte equal to the number of vertices (let us denote it by n)
- We have n sections (one for each vertex)
- Each section contains neighbours in clockwise order

Generating optimal 1-planar graphs

Planar Code

- Start with a single byte equal to the number of vertices (let us denote it by n)
- We have n sections (one for each vertex)
- Each section contains neighbours in clockwise order
- End of each section is determined by a 0 byte

5   340   350   14520   1530   2340

Definition 11
Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G' *isomorphic*, and write $G \simeq G'$, if there exists a bijection $\varphi : V \to V'$ with $(x, y) \in E \iff (\varphi(x), \varphi(y)) \in E'$ $\forall x, y \in V$. Such a map φ is called an *isomorphism*.

Definition 11
Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G' *isomorphic*, and write $G \simeq G'$, if there exists a bijection $\varphi : V \to V'$ with $(x, y) \in E \iff (\varphi(x), \varphi(y)) \in E' \ \forall x, y \in V$. Such a map φ is called an *isomorphism*.

Idea: Generate the optimal 1-planar graphs, pick one for each isomorphism class and reject all the others.

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:
    - Hash a graph G into a string,

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:
  - Hash a graph G into a string,
  - Compute all hash strings for graphs which are isomorphic to G,

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:
    - Hash a graph G into a string,
    - Compute all hash strings for graphs which are isomorphic to G,
    - Hash function is given by: build binary string by adding a 1 if there is an edge between two vertices (in order of vertex labeling); the hash string which is lexicographically the largest is called "Canonical Hash",

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:
  - Hash a graph G into a string,
  - Compute all hash strings for graphs which are isomorphic to G,
  - Hash function is given by: build binary string by adding a 1 if there is an edge between two vertices (in order of vertex labeling); the hash string which is lexicographically the largest is called "Canonical Hash",
  - Two graphs are isomorphic if their canonical hash is identical,

Generating optimal 1-planar graphs

- Canonical Graph Labeling (McKay)
- Core idea:
  - Hash a graph G into a string,
  - Compute all hash strings for graphs which are isomorphic to G,
  - Hash function is given by: build binary string by adding a 1 if there is an edge between two vertices (in order of vertex labeling); the hash string which is lexicographically the largest is called "Canonical Hash",
  - Two graphs are isomorphic if their canonical hash is identical,
  - McKay's algorithm is a search algorithm which finds the canonical hash faster.

Summary

Generating optimal 1-planar graphs

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,

Generating optimal 1-planar graphs

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,
- The quadrangulations must be 3-connected,

Generating optimal 1-planar graphs

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,
- The quadrangulations must be 3-connected,
- There are no optimal 1-planar graphs of order less than 8 and no of order 9,

Generating optimal 1-planar graphs

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,
- The quadrangulations must be 3-connected,
- There are no optimal 1-planar graphs of order less than 8 and no of order 9,
- Planar graphs can be represented using rotation systems,

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,
- The quadrangulations must be 3-connected,
- There are no optimal 1-planar graphs of order less than 8 and no of order 9,
- Planar graphs can be represented using rotation systems,
- We can efficiently determine the faces,

Summary

- The optimal 1-planar graphs can be obtained by generating the simple quadrangulations and adding the crossing diagonals in each face,
- The quadrangulations must be 3-connected,
- There are no optimal 1-planar graphs of order less than 8 and no of order 9,
- Planar graphs can be represented using rotation systems,
- We can efficiently determine the faces,
- Rejecting isomorphic copies can be done via nauty and traces

Generating optimal 1-planar graphs

- The tool plantri by Brinkmann et. al can efficiently generate all simple 3-connected quadrangulations,

Generating optimal 1-planar graphs

- The tool plantri by Brinkmann et. al can efficiently generate all simple 3-connected quadrangulations,
- It is open source.

Idea: Extend plantri in a way such that the crossing diagonals are added after the face expansions.

Problems:

Generating optimal 1-planar graphs

Problems:

- Plantri is written in a single *C* file containing approximately 20,000 lines of code,

Generating optimal 1-planar graphs

Problems:

- Plantri is written in a single C file containing approximately 20,000 lines of code,
- No publicly available unit tests,

Problems:

- Plantri is written in a single *C* file containing approximately 20,000 lines of code,
- No publicly available unit tests,
- Hence, there is a high risk that any code changes will break the algorithm without us noticing,

Generating optimal 1-planar graphs

Problems:

- Plantri is written in a single C file containing approximately 20,000 lines of code,
- No publicly available unit tests,
- Hence, there is a high risk that any code changes will break the algorithm without us noticing,
- Let us contact the author for advice.

Generating optimal 1-planar graphs

Gunnar Brinkmann <Gunnar.Brinkmann@ugent.be>
an mich

Englisch > Deutsch Nachricht übersetzen

Dear David Scholz,

[...]

The only hint I can give you about the code is: DO NOT CHANGE IT.

[...]

Best wishes,

Gunnar

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,
- We generated the simple quadrangulations within sage and added the crossing diagonals

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,
- We generated the simple quadrangulations within sage and added the crossing diagonals
- This had several advantages:

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,
- We generated the simple quadrangulations within sage and added the crossing diagonals
- This had several advantages:
    - Plantri is state of the art for planar graph generation,

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,
- We generated the simple quadrangulations within sage and added the crossing diagonals
- This had several advantages:
  - Plantri is state of the art for planar graph generation,
  - Isomorphism rejection is build in (nauty and traces),

Generating optimal 1-planar graphs

- There is a Sage integration of plantri,
- We generated the simple quadrangulations within sage and added the crossing diagonals
- This had several advantages:
    - Plantri is state of the art for planar graph generation,
    - Isomorphism rejection is build in (nauty and traces),
    - Our conjecture-making program is also available as a sage package (hence we are staying in the same ecosystem).

Generating optimal 1-planar graphs

Plantri running times:

Graph count	Vertex count	CPU time (seconds)
1	8	0.00
1	10	0.00
1	11	0.00
3	12	0.00
3	13	0.00
11	14	0.00
18	15	0.00
58	16	0.00
139	17	0.00
451	18	0.00
1326	19	0.00
4461	20	0.01
14554	21	0.04
49957	22	0.15
171159	23	0.49
598102	24	1.70
2098675	25	5.87
7437910	26	20.67
26490072	27	72.51
94944685	28	255.02
341867921	29	904.37
1236864842	30	3253.92

Generating optimal 1-planar graphs

Prohibiting printing to stdout:

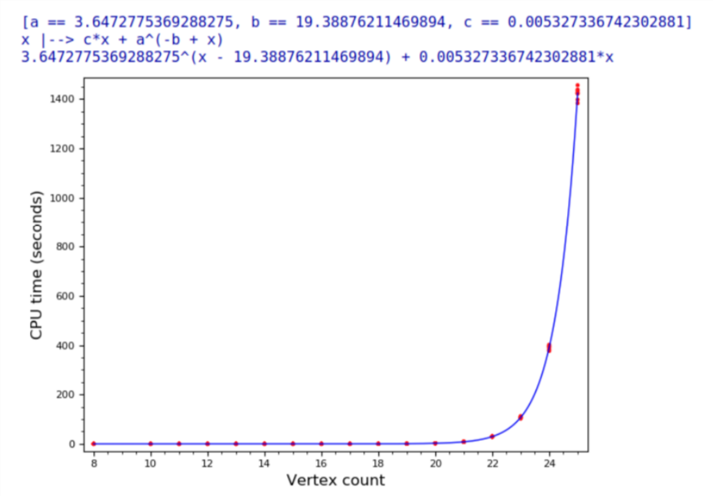
Graph count	Vertex count	CPU time (seconds)
1	8	0.00
1	10	0.00
1	11	0.00
3	12	0.00
3	13	0.00
11	14	0.00
18	15	0.00
58	16	0.00
139	17	0.00
451	18	0.00
1326	19	0.00
4461	20	0.01
14554	21	0.04
49957	22	0.13
171159	23	0.43
598102	24	1.47
2098675	25	5.06
7437910	26	17.70
26490072	27	61.60
94944685	28	216.31
341867921	29	766.02
1236864842	30	2726.30

Generating optimal 1-planar graphs

Sage:

Graph count	Vertex count	CPU time (seconds)
1	8	0.01
1	10	0.01
1	11	0.01
3	12	0.01
3	13	0.01
11	14	0.01
18	15	0.01
58	16	0.02
139	17	0.04
451	18	0.11
1326	19	0.33
4461	20	1.15
14554	21	3.94
49957	22	14.13
171159	23	50.79
598102	24	187.32
2098675	25	703.74
7437910	26	2491.38
26490072	27	9315.46
94944685	28	33326.80
341867921	29	127407.80
1236864842	30	–

Generating optimal 1-planar graphs



Automated conjecture-making

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?
- Bondy collected some basic properties

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?
- Bondy collected some basic properties
  - Simplicity: Short and easy to understand

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?
- Bondy collected some basic properties
  - Simplicity: Short and easy to understand
  - Surprise: Establishes a surprising connection between seemingly unrelated concepts

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?
- Bondy collected some basic properties
  - Simplicity: Short and easy to understand
  - Surprise: Establishes a surprising connection between seemingly unrelated concepts
  - Generality: True for a variety of objects (a whole class of graphs)

Automated conjecture-making

- A conjecture is a formal statement which is initially open to formal proofs
- Without keeping an automatism in mind: What constitutes a good conjecture?
- Bondy collected some basic properties
  - Simplicity: Short and easy to understand
  - Surprise: Establishes a surprising connection between seemingly unrelated concepts
  - Generality: True for a variety of objects (a whole class of graphs)
  - Fecundity: Proof attempts led to new concepts or new proof techniques.

Automated conjecture-making

Conjectures can take many forms.

Automated conjecture-making

Conjectures can take many forms.

- Property-based conjectures ("Every 4-connected planar graph is hamiltonian")

Automated conjecture-making

Conjectures can take many forms.

- Property-based conjectures ("Every 4-connected planar graph is hamiltonian")
- Conjectures in form of inequalities: $I \leq J$, $I \leq J + K$, $I + J \leq K + L$ with I, J, K, L being invariants ($\chi \leq \Delta + 1$ (Brook's theorem))

Automated conjecture-making

A program can produce an endless stream of conjectures in form of inequalities.

Automated conjecture-making

A program can produce an endless stream of conjectures in form of inequalities.

- How can we decide which statements are significant and which are not?

Automated conjecture-making

A few formalities:

Automated conjecture-making

A few formalities:

- Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be some computable invariants (in form of functions) of a mathematical object G,

Automated conjecture-making

A few formalities:

- Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be some computable invariants (in form of functions) of a mathematical object G,
- We define some operations $f_1, f_2, \cdots, f_r$ of some algebraic system,

Automated conjecture-making

A few formalities:

- Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be some computable invariants (in form of functions) of a mathematical object G,
- We define some operations $f_1, f_2, \cdots, f_r$ of some algebraic system,
- For instance a term is given by $f(\alpha_i, \alpha_j)$ (which is a new invariant),

Automated conjecture-making

A few formalities:

- Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be some computable invariants (in form of functions) of a mathematical object G,
- We define some operations $f_1, f_2, \cdots, f_r$ of some algebraic system,
- For instance a term is given by $f(\alpha_i, \alpha_j)$ (which is a new invariant),
- If s and t are terms, the expression $t \leq s$ is a statement,

Automated conjecture-making

A few formalities:

- Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be some computable invariants (in form of functions) of a mathematical object G,
- We define some operations $f_1, f_2, \cdots, f_r$ of some algebraic system,
- For instance a term is given by $f(\alpha_i, \alpha_j)$ (which is a new invariant),
- If s and t are terms, the expression $t \leq s$ is a statement,
- We call a statement which holds for every object G in our data set a *candidate*.

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,
- Suppose we have a finite set $\mathcal{C}$ of pre-existing conjectures for the invariant α of interest in similar form (e.g. $\alpha \leq u_1, \cdots,$ $\alpha \leq u_k$)

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,
- Suppose we have a finite set $\mathcal{C}$ of pre-existing conjectures for the invariant α of interest in similar form (e.g. $\alpha \leq u_1, \cdots,$ $\alpha \leq u_k$)
- $\mathcal{G}$ and $\mathcal{C}$ might be initially empty,

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,
- Suppose we have a finite set $\mathcal{C}$ of pre-existing conjectures for the invariant α of interest in similar form (e.g. $\alpha \leq u_1, \cdots,$ $\alpha \leq u_k$)
- $\mathcal{G}$ and $\mathcal{C}$ might be initially empty,
- Now, let $\alpha \leq s$ be a new conjecture made by our program, then we perform the following steps

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,
- Suppose we have a finite set $\mathcal{C}$ of pre-existing conjectures for the invariant α of interest in similar form (e.g. $\alpha \leq u_1, \cdots,$ $\alpha \leq u_k$)
- $\mathcal{G}$ and $\mathcal{C}$ might be initially empty,
- Now, let $\alpha \leq s$ be a new conjecture made by our program, then we perform the following steps
  - Truth test: $\alpha \leq s$ is true for all $G \in \mathcal{G}$

Automated conjecture-making

- Suppose we have a finite set $\mathcal{G}$ of objects,
- Suppose we have a finite set $\mathcal{C}$ of pre-existing conjectures for the invariant α of interest in similar form (e.g. $\alpha \leq u_1, \cdots,$ $\alpha \leq u_k$)
- $\mathcal{G}$ and $\mathcal{C}$ might be initially empty,
- Now, let $\alpha \leq s$ be a new conjecture made by our program, then we perform the following steps
    - Truth test: $\alpha \leq s$ is true for all $G \in \mathcal{G}$
    - Significance test: at least one $G' \in \mathcal{G}$ for which $s(G') < t(G'), \forall t \in \mathcal{C}$.

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
  - Initialize a set of objects with cardinality at least 1,

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
    - Initialize a set of objects with cardinality at least 1,
    - Generate a stream of conjectures

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
    - Initialize a set of objects with cardinality at least 1,
    - Generate a stream of conjectures
    - Each conjecture must pass the truth and significance test before added to the list of conjectures,

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
    - Initialize a set of objects with cardinality at least 1,
    - Generate a stream of conjectures
    - Each conjecture must pass the truth and significance test before added to the list of conjectures,
    - Remove insignificant conjectures (check if a newly added conjecture makes another one insignificant)

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
    - Initialize a set of objects with cardinality at least 1,
    - Generate a stream of conjectures
    - Each conjecture must pass the truth and significance test before added to the list of conjectures,
    - Remove insignificant conjectures (check if a newly added conjecture makes another one insignificant)
    - Search for counterexamples

Automated conjecture-making

The truth and significance test are called the *Fatjlowicz-Dalmatian Heuristic*.

- Implemented in the program *conjecturing* by Larson and Van Cleemput.
  - Initialize a set of objects with cardinality at least 1,
  - Generate a stream of conjectures
  - Each conjecture must pass the truth and significance test before added to the list of conjectures,
  - Remove insignificant conjectures (check if a newly added conjecture makes another one insignificant)
  - Search for counterexamples
  - Repeat conjecture generation.

Automated conjecture-making

Typical output:

1.	independence_number(x)	<=	ceil(arcsinh(log(size(x))))
2.	independence_number(x)	<=	1/2*min_degree(x)
3.	independence_number(x)	<=	-min_degree(x) + order(x)
4.	independence_number(x)	<=	floor(arcsinh(order(x)))
5.	independence_number(x)	<=	-minimum(num_skel_f(x), 1/4*min_degree(x)^2) + order(x)
6.	independence_number(x)	<=	crossing_number(x)/min_degree(x)
7.	independence_number(x)	<=	-floor(tan(size(x))) + num_skel_f(x)
8.	independence_number(x)	<=	ceil(log(crossing_number(x)))
9.	independence_number(x)	<=	floor(crossing_number(x)/min_degree(x))
10.	independence_number(x)	<=	2*(min_degree(x) - num_skel_f(x) + 1)^2
11.	independence_number(x)	<=	-size(x)/(max_degree(x) - order(x))
12.	independence_number(x)	<=	sqrt(max_degree(x) + order(x) - 1)
13.	independence_number(x)	<=	-1/4*max_degree(x) + 1/2*order(x)
14.	independence_number(x)	<=	-min_degree(x) + order(x)
15.	independence_number(x)	<=	-diameter(x) + 1/2*order(x)

Automated conjecture-making

Optimizations

Optimizations

- Precalulate graphs and their invariants and store them in a database,

Automated conjecture-making

Optimizations

- Precalulate graphs and their invariants and store them in a database,
- Use known bounds from literature and store them in a database (Larson actually already published one),

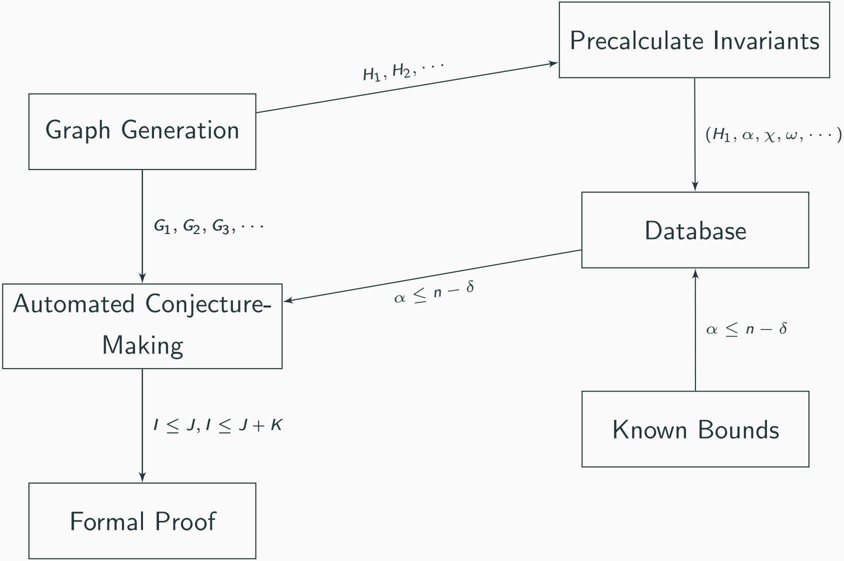
Optimizations

- Precalulate graphs and their invariants and store them in a database,
- Use known bounds from literature and store them in a database (Larson actually already published one),
- Use the project "House of Graphs" (database of the Ghent University containing large number of graphs),

Optimizations

- Precalulate graphs and their invariants and store them in a database,
- Use known bounds from literature and store them in a database (Larson actually already published one),
- Use the project "House of Graphs" (database of the Ghent University containing large number of graphs),
- Speed up the generation process.

Automated conjecture-making



Discussion

Conclusions

- We introduced the optimal 1-planar graphs,

Conclusions

- We introduced the optimal 1-planar graphs,
- We have shown that they can be obtained by generating the simple quadrangulations and add the crossing diagonals in each face,

Conclusions

- We introduced the optimal 1-planar graphs,
- We have shown that they can be obtained by generating the simple quadrangulations and add the crossing diagonals in each face,
- We implemented a generation algorithm,

Conclusions

- We introduced the optimal 1-planar graphs,
- We have shown that they can be obtained by generating the simple quadrangulations and add the crossing diagonals in each face,
- We implemented a generation algorithm,
- We introduced the concept of automated conjecture-making and the Fajtlowicz-Dalmatian Heuristic

Conclusions

- We introduced the optimal 1-planar graphs,
- We have shown that they can be obtained by generating the simple quadrangulations and add the crossing diagonals in each face,
- We implemented a generation algorithm,
- We introduced the concept of automated conjecture-making and the Fajtlowicz-Dalmatian Heuristic
- We presented the program *conjecturing* and used it for calculating upper bounds for the independence number of the optimal 1-planar graphs.

Thank You!

1. Christopher Auer et al. "1-Planarity of Graphs with a Rotation System." In: J. Graph Algorithms Appl. 19.1 (2015), pp. 67–86.
2. David Barnette. "On generating planar graphs". In: Discrete Mathematics 7.3-4 (1974), pp. 199–208.
3. V. Berhard. Zur Morphologie der Polyeder. Teubner, 1891.
4. R. Bodendiek, H. Schumacher, and K. Wagner. "Uber 1-optimale Graphen". In: Mathematische Nachrichten 117.1 (1984), pp. 323–339.
5. Adrian Bondy. "Beautiful conjectures in graph theory." In: Eur. J. Comb. 37 (2014), pp. 4–23.

6. Gunnar Brinkmann and Brendan McKay. Guide to using plantri (version 5.0). url: https://users.cecs.anu.edu.au/ bdm/plantri/plantri-guide.txt (visited on 11/10/2019).
7. Gunnar Brinkmann and Brendan McKay. plantri and fullgen. url: http://users. cecs.anu.edu.au/ bdm/plantri/ (visited on 12/26/2019).
8. Gunnar Brinkmann, Brendan D. McKay, et al. "Fast generation of planar graphs". In: MATCH Commun. Math. Comput. Chem 58.2 (2007), pp. 323–357.

References iii

9. Gunnar Brinkmann et al. “Generation of simple quadrangulations of the sphere”. In: Discrete mathematics 305.1-3 (2005), pp. 33–54.
10. Gunnar Brinkmann et al. House of Graphs: a database of interesting graphs. url: http://hog.grinvin.org (visited on 12/26/2019).
11. Igor Fabrici and Tomas Madaras. “The structure of 1-planar graphs”. In: Discrete Mathematics 307.7-8 (2007), pp. 854–865.
12. Siemion Fajtlowicz. “Toward fully automated fragments of graph theory”. In: (2003).

13. Pierre Hansen et al. "What Forms Do Interesting Conjectures Have in Graph Theory?" In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science 69 (2005).
14. Mohammadreza Jooyandeh. "Recursive Algorithms for Generation of Planar Graphs". PhD thesis. The Australian National University, 2014.
15. Craig E. Larson. "A survey of research in automated mathematical conjecture- making". In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science 69 (2005), p. 297.

References v

16. Craig E. Larson and Nico Van Cleemput. "Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited". In: Artificial Intelligence 231 (2016), pp. 17–38.
17. Brendan McKay. graph formats. url: http://users.cecs.anu.edu.au/ bdm/ data/formats.html (visited on 12/26/2019).
18. Brendan D. McKay and Adolfo Piperno. "Practical graph isomorphism, II". In: Journal of Symbolic Computation 60 (2014), pp. 94–112.
19. James Munkres. Topology. Pearson Education, 2014.

20. Gerhard Ringel. “Ein sechsfarbenproblem auf der Kugel”. In: Abhandlungen aus dem Mathematischen Seminar der Universitat Hamburg. Vol. 29. 1. Springer. 1965, pp. 107–117.
21. Yusuke Suzuki. “Optimal 1-planar graphs which triangulate other surfaces”. In: Discrete Mathematics 310.1 (2010), pp. 6–11.
22. Marlen Rutsch. ”An updated annotated bibliography and new results on 1-planarity”. Master’s Thesis. TH Koeln 2019.