Karlsruhe Institute of Technology

Partitioning Geometric Graphs into Plane Subgraphs

Bachelor Thesis of

Jonathan Benedikt Dransfeld

At the Department of Informatics
Institute of Theoretical Informatics

Reviewers:	PD Dr. Torsten Ueckerdt
	Juniorprofessor Dr. Thomas Bläsius
Advisors:	Paul Jungeblut
	Laura Merker

Time Period: 18th October 2021 – 18th February 2022

Statement of Authorship

Karlsruhe, February 18, 2022

Abstract

A geometric graph is a set of points in the plane with connecting straight-line segments. This thesis examines the problem of partitioning geometric graphs into subgraphs without intersecting segments as presented in the CG:SHOP 2022 competition. This problem is equivalent to solving the coloring problem on the intersection graph of these geometric graphs.

We show that this problem is NP-complete, even with a limited number of slopes and maximum partitions. Further, we analyze the competition instances for properties that aid in coloring the intersection graphs, and evaluate the run-time and colors required for known coloring algorithms. Additionally, we introduce a new sequential coloring algorithm, which orders the vertices of the intersection graph by the slopes of their corresponding segment in the geometric graph. This algorithm requires fewer colors than the well-known DSATUR algorithm on a majority of the competition instances.

Deutsche Zusammenfassung

Ein geometrischer Graph ist eine Menge von Punkten in der Ebene und eine Menge von gradlinigen Segmenten, welche diese Punkte verbinden. Diese Bachelorarbeit beschäftigt sich mit dem Problem geometrische Graphen in Subgraphen zu unterteilen, in denen sich keine zwei Segmente schneiden. Dieses Problem wurde im CH:SHOP 2022 Wettbewerb vorgestellt. Ein äquivalentes Problem ist das Färbungsproblem auf den Schnittgraphen dieser geometrischen Graphen.

Wir zeigen, dass dieses Problem NP-vollständig ist, auch wenn man die Anzahl der Steigungen der Segmente und die maximale Anzahl an Partitionen begrenzt. Wir untersuchen die Wettbewerbsinstanzen auf Eigenschaften, welche das Färbungsproblem auf den Schnittgraphen erleichtern, und bewerten die Laufzeit und Anzahl benötigter Farben von bekannten Färbungsalgorithmen. Zudem stellen wir einen neuen sequentiellen Färbungsalgorithmus vor, welcher die Knoten des Schnittgraphens nach den Steigungen der zugehörigen Segmente im geometrischen Graphen sortiert. Dieser Algorithmus benötigt weniger Farben für die meisten Wettbewerbsinstanzen als der bekannte DSATUR-Algorithmus.

Contents

1. Introduction

This thesis examines the problem of partitioning geometric graphs into plane subgraphs. A geometric graph is a set of points in the two-dimensional plane, connected by straight line segments. An example of geometric graph is shown in Figure 1.1. The colors of the lines represent a partition, where each color represents one partition. Two lines of the same color do not intersect, making these partitions plane.

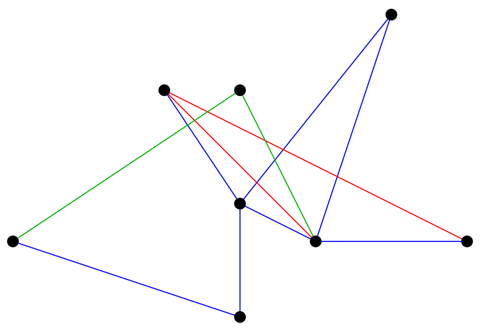

Figure 1.1.: An example partition of geometric graph into three subgraphs.

This partitioning problem is equivalent to the coloring problem on the intersection graph of the geometric graph. This leads us to solving a coloring problem in this thesis.

Motivation

The main motivation for this thesis is that the problem of partitioning geometric graphs into plane subgraphs was posed in the "CG:SHOP 2022" (Computational Geometry: Solving Hard Optimization Problems) competition. This annual competition by the TU Braunschweig focuses on one optimization problem in computational geometry and provides a set of problem instances for the participants to solve as optimally as possible. The problem statement, competition instances, and further information can be found on the official CG:SHOP 2022 website https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2022/.

This problem is closely related to other theoretical problems of interest. Interval graphs, a special case of geometric graphs where all segments endpoints lie on a line, are a natural representation for scheduling, gene sequencing, and combinatorial problems. A short survey on interval graphs, related graphs classes, and their applications is Charles' paper [Gol85].

The edge coloring problem for planar graphs can be transformed into our problem by extending each edge of plane drawing with straight lines slightly, making this another special case of our problem.

Related work

A special case of the problem of partitioning geometric graphs into plane subgraphs is a version where all segment endpoints are on a circle. This problem is known to be solvable in polynomial time [GJMP80]. Another special case, the edge coloring of planar graphs named in the motivation section, is discussed in [CN90].

Other NP-hard problems on segment intersection graphs are already studied, like the clique problem [CCL13, KN90], the independent set problem [KN90] and the problem of recognizing segment intersection graphs [Kra91].

Contribution

This thesis features three key contributions.

The problem of partitioning geometric graphs into plane subgraphs is known to be NP-complete. We add another proof of NP-completeness, which also contributes a new proof which shows that this problem is already NP-complete when limiting the number of slopes to four, limiting the number of partitions to three, and prohibiting parallel segments in the geometric graph to intersect.

We provide an analysis of the 225 competition instances. This section mainly features noteworthy properties of the intersection graphs that help us to color them. Additionally, we note properties of the geometric graphs representing these intersection graphs that help solve the coloring problem on the intersection graphs.

Last, we introduce new coloring algorithms for intersection graphs with known geometric graph representations and compare them to well-known coloring algorithms.

Outline

The content of this thesis is structured in four chapters.

Chapter 2 introduces the preliminaries. First, we introduce basic notations and the graph classes we examine in this thesis. Then, we formally state the problem of partitioning geometric graphs into plane subgraphs we investigate in this thesis, as well as an important Lemma about the invariance of their intersection graphs under certain transformations.

Chapter 3 shows the NP-completeness of the problem we solve, as well as with a limited number of slopes, fixed choices of slopes, and a limited number of maximum partitions.

Chapter 4 provides the instance analysis, introducing the notable differences between our graphs and general graphs. This motivates the investigation of this problem further due to it not simply being a repetition of the research on general graphs. The properties found in this chapter are also used in the following chapter to motivate and assist the coloring algorithms, and may be used as a starting point for further research with different approaches.

In Chapter 5, we introduce the algorithms we use to solve the problem of partitioning geometric graphs into plane subgraphs, and evaluate them.

2. Preliminaries

In this chapter, we introduce the main problem we examine, partitioning geometric graphs into plane subgraphs, as well as the definitions and notations used in this thesis.

A *graph* is a tuple of a set V called *vertices* and a set E called *edges*, with $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Graphs are commonly denoted by $G = (V, E)$ and the edge $\{u, v\}$ is denoted by uv. A *random graph* $\mathcal{G}_{n,p}$ is graph with n vertices where each edge exists independently with probability p.

For a subset of edges $V' \subset V$, we call the graph (V', E') with $\{uv \in E \mid u \in V' \land v \in V\}$ the *induced subgraph* of V'. The notation $G - v$ for some vertex $v \in V$ refers to the induced subgraph with $V' = V \setminus \{v\}$.

The notation $\mathbb{Z}_n = \{1, \ldots, n\}$ refers to the first n natural numbers, from 1 to n. An *order* σ of a finite set S is a bijective map from the first $|S|$ natural numbers to the elements of the set $\sigma : \mathbb{Z}_{|S|} \to S$.

2.1. Intersection graphs

A graph $R = (P, S)$ where all vertices are points in a two-dimensional space, $P \subseteq \mathbb{R}^2$, is called a *geometric graph*. The edges of a geometric graph are also called *segments* and are represented by straight lines between their endpoints.

Two segments p_1p_2 and p_3p_4 *intersect* when they have a common points except a common endpoint. Let $\overline{pq} = \{p + t \cdot (q - p) \mid t \in [0, 1]\}$ be the closed straight line between the points p and q. The two segments p_1p_2 and p_3p_4 intersect if the following is true:

$$(\overline{p_1p_2} \cap \overline{p_3p_4}) \setminus (\{p_1, p_2\} \cap \{p_3, p_4\}) \neq \emptyset$$

The class of all intersection graphs of geometric graphs is called Seg as defined in section 1.2 of [KM94]. Let R be the intersection graph of the segments generated by G. Then R is called the *representation* of G.

The *slope* of a segment is the minimal angle of clockwise rotation about its center required to make that segment horizontal. For two segments with slopes d_1, d_2, we define their *intersection angle* as $\min(|d_2 - d_1|, \pi - |d_2 - d_1|)$. A horizontal segment has a slope of 0. Every segment has a slope in $[0, \pi)$, and every intersection angle is in $[0, \pi/2)$. [KM94] defines the following subclasses of Seg which limit the allowed slopes of the segments in the representation. The graph class k-Dir$(d_1, \dots, d_k)$ is defined as all graphs that have a representation where only the slopes $d_1, \dots, d_k$ are used. The graph class k-Dir is defined as

$$\text{k-Dir} = \bigcup_{(d_1,\dots,d_k)\in[0,\pi)^k} \text{k-Dir}(d_1, \dots, d_k)$$

The class Pure-k-Dir$(d_1, \dots, d_k)$ is the set of all graphs that have an intersection graph with only the slopes $d_1, \dots, d_k$, where segments with the same slope do not intersect. The graph class Pure-k-Dir is defined as

$$\text{Pure-k-Dir} = \bigcup_{(d_1,\dots,d_k)\in[0,\pi)^k} \text{Pure-k-Dir}(d_1, \dots, d_k)$$

Further, for any $k \in \mathbb{N}$:

$$\begin{aligned} \text{k-Dir} &\subseteq \text{(k + 1)-Dir} \\ \text{Pure-k-Dir} &\subseteq \text{Pure-(k + 1)-Dir} \\ \text{Pure-k-Dir} &\subseteq \text{k-Dir} \\ \text{k-Dir} &\subseteq \text{SEG} \end{aligned}$$

2.2. Partitions and plane graphs

Now that we have established the graph classes relevant for this thesis, we define the prerequisites and the problem itself of partitioning geometric graphs into plane subgraphs in this section.

A representation is called *plane* if no two segments intersect. An equivalent definition is that the intersection graph contains no edges. A geometric graph is called plane if the representation constructed from it is plane.

The sets $M_1, M_2, \dots, M_k$ are called a *partition* of M if their union is M and the sets are pairwise disjoint:

$$\bigcup_{i=1}^{k} M_i = M$$
$$\forall i, j \in \{1, \dots, k\}, i \neq j : M_i \cap M_j = \emptyset$$

Definition 2.1 (Partitioning geometric graphs into plane subgraphs)**.** *Let $G = (V, E), V \subseteq \mathbb{R}^2$ be a geometric graph. The goal is to partition the set E into $E_1, \dots, E_n$ for some $k \in \mathbb{N}$, such that all the graphs $G_i = (V, E_i), k \in \{1, \dots, n\}$ are plane.*

The target to minimize is n, the number of partitions required.

A *coloring* of a graph $G = (V, E)$ with n colors is a map $c : V \to \mathbb{N}$ where adjacent vertices do not have the same color, formally $vw \in E \implies c(v) \neq c(w)$.

If $c : V' \to \mathbb{Z}_n$ is a coloring of the intersection graph $G = (V', E')$, then defining the partitioning of the geometric graph $G = (V, E)$ as $E_i = \{v \in V' \mid c(v) = i\}$ defines a plane partitioning. Throughout this thesis, we do not attempt to partition geometric graphs directly, but focus on the equivalent problem of coloring their respective intersection graphs.

2.3. Invariance of intersection graphs

A *translation* by a vector $v \in \mathbb{R}^2$ is the function

$$t_v : \mathbb{R}^2 \to \mathbb{R}^2 : p \mapsto p + v$$

The inverse of t_v is t_{-v}.

Lemma 2.2. *Let $f : \mathbb{R}^2 \to \mathbb{R}^2$ be an invertible function that keeps straight lines. The intersection graphs of the representations (P, S) and $(f(P), S)$ are the same.*

Proof. Let (V, E) be the intersection graph of (P, S), and (V', E') the one of $(f(P), S)$. Because of $V = S$ and $V' = S$, it follows that $V = V'$.

Let $vw \in E$ be an edge. Then there is a point $p \in \mathbb{R}^2$ such that p lies on both segments, v and w. It follows that $f(p)$ is both on $f(v)$ and $f(w)$, Therefore, $vw \in E'$ and $E \subseteq E'$. The proof for $vw \in E$ and $E' \subseteq E$ is analogous using the inverse function f^{-1}. $\square$

All translation t_v and invertible linear transformations fulfill the conditions of Lemma 2.2.

3. NP-completeness of Seg-k-Color

In this chapter, we show that the problem SEG-3-COLOR is NP-complete. Additionally, we proof that the problem is already NP-complete on smaller subclasses of graphs. For an introduction to NP-completeness, see Karp's paper [Kar72].

Let $\mathcal{G}$ be a graph class and $k \in \mathbb{N}$ a natural number. We define the k-coloring problem on this graph class as follows:

Definition 3.1 ($\mathcal{G}$-k-COLOR)**.** *Given a graph $G \in \mathcal{G}$, is it possible to properly color G with at most k colors?*

The final goal of this chapter is to proof the following Theorem 3.2:

Theorem 3.2. *SEG-K-COLOR is NP-complete.*

To simplify the proofs of NP-completeness, both for Theorem 3.2 and subclasses of Seg, we introduce the following two Lemmas.

Lemma 3.3. *For all graph classes $\mathcal{G}$ and natural numbers $k \in \mathbb{N}$, the problem $\mathcal{G}$-k-COLOR is in NP.*

Proof. Let $c : V \to \mathbb{Z}_k \cup \{\perp\}$ be a non-deterministically generated coloring of the graph $G = (V, E)$. Checking that the coloring is complete, which requires checking $c(v) \neq \perp$ for every vertex $v \in V$, requires $\mathcal{O}(|V|)$ time. Checking that the coloring uses at most k colors, which requires checking $c(v) \leq k$ for every vertex $v \in V$, requires $\mathcal{O}(|V|)$ time. Checking that the coloring uses at most k colors, which requires checking $c(u) \neq c(v)$ for every edge $uv \in E$, requires $\mathcal{O}(|E|)$ time. The total time required is in $\mathcal{O}(|V| + |E|)$. □

Lemma 3.4. *Let $\mathcal{G}, \mathcal{G}'$ be two graph classes with $\mathcal{G}' \subseteq \mathcal{G}$, and $k \in \mathbb{N}$. If $\mathcal{G}'$-k-COLOR is NP-complete, then $\mathcal{G}$-k-COLOR is NP-complete as well.*

Proof. $\mathcal{G}$-k-COLOR is in NP following lemma 3.3.

Let f be the polynomial transformation from some known NP-complete problem NP-PROBLEM to $\mathcal{G}'$-k-COLOR. An instance of $\mathcal{G}'$-k-COLOR only consists out of a graph $G \in \mathcal{G}'$, which is also a graph in $\mathcal{G}$. This means that f is also a polynomial transformation from NP-PROBLEM to $\mathcal{G}$-k-COLOR. Therefore, $\mathcal{G}$-k-COLOR is NP-hard. □

Following Lemma 3.4, there are multiple other proofs of Theorem 3.2 Planar graphs, the class of all graphs that have a plane drawing, are a subset of all segment intersection graphs [CG09]. The NP-completeness of 3-Color is already proven on multiple subclasses of planar graphs, like hamiltonian planar [Cav19], 4-regular planar [Dai80], and planar with maximum degree 4 [GJS74] graphs.

3.1. NP-completeness of Pure-4-Dir-3-Color

In the section, we show that the problem Pure-4-Dir(d_1, d_2, d_3, d_4)-3-Color is NP-complete for every set of four slopes $(d_1, d_2, d_3, d_4) \in [0, \pi)^4$. We first proof this for a fixed set of four slopes and then generalize this proof to every set of four slopes

NP-hardness of Pure-4-Dir$(0, \pi/4, \pi/2, 3\pi/4)$-3-Color

In this section, we show the NP-hardness of Pure-4-Dir$(0, \pi/4, \pi/2, 3\pi/4)$-3-Color with the polynomial transformation from the known NP-complete problem 3-SAT [Kar72] to Pure-4-Dir$(0, \pi/4, \pi/2, 3\pi/4)$-3-Color. This is the proof we later generalize to the proof that the problem is NP-complete for any set of four slopes.

The goal is to construct a graph in Pure-4-Dir that is 3-colorable if and only if the variables U of the 3-SAT instance (U, C) have an assignment of boolean values, fulfilling all clauses C.

To prove that the graph is in Pure-4-Dir, we construct the representation instead of the intersection graph. The slopes used in the representation are $\{0, \pi/4, \pi/2, 3\pi/4\}$, and no two parallel segments intersect. Therefore, the intersection graph is in Pure-4-Dir$(0, \pi/4, \pi/2, 3\pi/4)$.

For the construction of this geometric graph, we use multiple small collections of segments called *gadget*. For each gadget, we define a *bounding box*. This is a rectangle with its sides parallel to the x-axis, a horizontal axis pointing right, and the y-axis, a vertical axis pointing up. The bottom left corner of the bounding box is always at the point $(0, 0)$ and its top right corner at (w, h), which we state for each gadget.

A gadget contains a segment if the segment and the gadget's bounding box intersect. For each gadget, we note its endpoints, slope, and its intersections with other segments.

The construction of the representation requires the following gadgets:

- A *base gadget*. This gadget is required once in the construction of the final graph for other gadgets to function. The goal of this gadget is to create three segments with pairwise differing colors, the color of one representing true, one color representing false, and one not representing a truth value. Additionally, the gadget has a vertical segment with the neutral color that extend downward as far as needed and two horizontal segments with the colors neutral and false that can extend as far right as needed. These three extra segments assist in the function of other gadgets later on.

  The bounding box of the base gadget spans from $(0, 0)$ to $(5, 10)$. It has the following five segments:

  - The *variable neutral segment* with endpoints $(1.5, 0.5)$ and $(1.5, y_{e2}), y_{\text{end}} > 10$. It has slope 0.
  - The *clause false segment* with endpoints $(0.5, 1.5)$ and $(x_{e2}, 1.5), x_{\text{end}} > 5$. It has slope $\pi/2$. It intersects the variable neutral segment in the point $p_1 = (1.5, 1.5)$.
  - The *base true segment* with endpoints $(1, 1)$ and $(3, 3)$. It has slope $3\pi/4$. It intersects the clause false segment and the variable neutral segment in the point p_1.

- The *base false segment* with endpoints $(1, 4)$ and $(3, 2)$. It has slope $\pi/4$. It intersects with the variable neutral segments in the point $p_2 = (1.5, 1.5)$ and with the base true segment in point $p_3 = (2.5, 2.5)$.

- The *clause neutral segment* with endpoints $(2, 2.5)$ and $(x_{e3}, 2.5), x_{e3} > 5$. It has slope 0. Its right endpoint is not fixed by the gadget. It intersects the base false segment and the base true segment in p_3.

The base gadget is shown in Figure 3.1. The segments in this figure are already colored. In this and the following figures of gadgets, the colors green, red and blue represent true, false and neutral, respectively.

Lemma 3.5. *Let c be a valid 3-coloring of the base gadget. We call the color of the base true segment the* true color*, the color of the base false segment the* false color*, and the color of the variable neutral segment the* neutral color*.*

These three colors are pairwise different.

Proof. The three segments defining the colors intersect pairwise. The base true and the variable neutral segment intersect in p_1, therefore, the true and neutral color are different. The base false and the variable neutral segment intersect in p_2, therefore, the false and neutral color are different. The base true and the base false segment intersect in p_3, therefore, the true and false color are different. □

We use the names given to the colors in Lemma 3.5 from now on for all gadgets.

Lemma 3.6. *Following the coloring c of Lemma 3.5, the base clause neutral segment has the neutral color and the clause false segment has the false color.*

Proof. The clause neutral segment intersects both the base true and base false segment in p_3, leaving the neutral color as the only possible one. The clause false segment intersects both the base true and variable neutral segment in p_1, leaving the false color as the only possible one. □

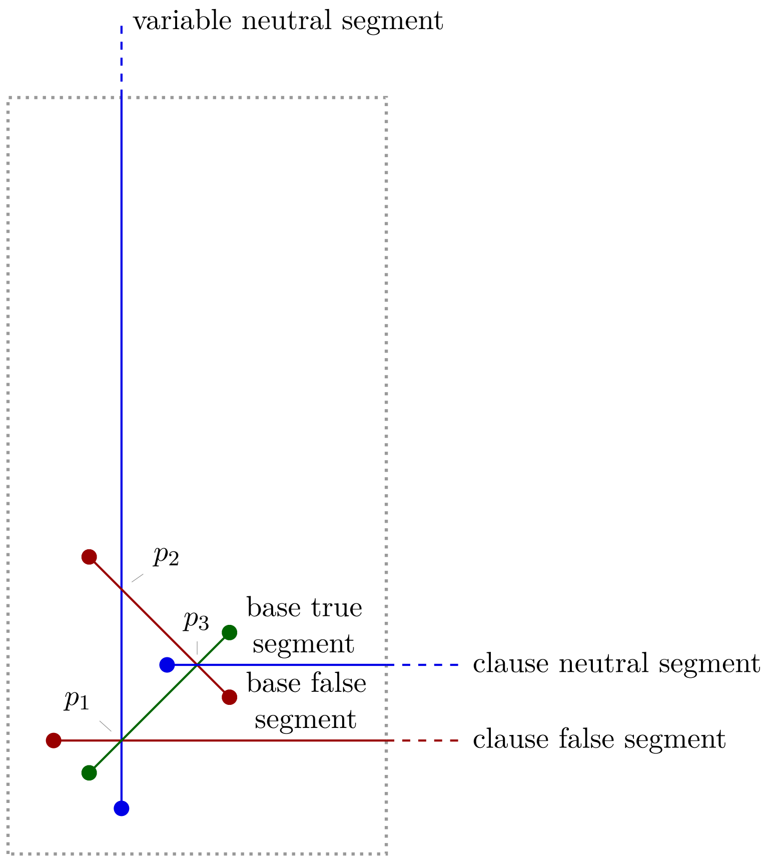

Figure 3.1.: The base gadget. All fixed endpoints are marked with dots. A dashed line at the end of a segment indicates that there is no fixed end to it, and it may be extended.

- A *variable gadget*. This gadget is placed once for every variable $x \in U$ in the 3-SAT instance. It provides two segments, one of which has the color true and one has the color false, assuming the base gadget is colored as described above. The color of these two segments represent the values of the literals x and $\bar{x}$. Which one has the color true and which the color false is not predetermined by the way the variable gadget is constructed. This gadget may use the variable neutral segment from the base gadget, but it must be able to extend through the gadget to be available for other variable gadgets below this one.

  The bounding box of the variable gadget spans from $(0, 0)$ to $(5, 10)$. It contains the following five segments:

  – The variable neutral segment, which was already defined in the base gadget. It passes through every instance of this gadget, from $(1.5, 10)$ to $(1.5, 0)$. Neither of its endpoints are in this gadget.

  – The $\bar{x}$ *exclusive segment* with endpoints $(1, 4.5)$ and $(4.5, 8)$. It has slope $3\pi/4$, and intersects the variable neutral segment in $p_4 = (1.5, 5)$.

  – The x *exclusive segment* with endpoints $(1, 5.5)$ and $(4.5, 2)$. It has slope $\pi/4$. It intersects the $\bar{x}$ exclusive segment and the variable neutral segment in the point p_4.

  – The *variable literal x segment* with endpoints $(1, 7.5)$ and $(10, 7.5)$. It has slope 0. It intersects the variable neutral segment in $p_5 = (1.5, 7.5)$ and the $\bar{x}$ exclusive segment in $p_6 = (4, 7.5)$.

- The *variable literal $\bar{x}$ segment* with endpoints $(1, 2.5)$ and $(10, 2.5)$. It has slope 0. It intersects the variable neutral segment in $p_5 = (1.5, 2.5)$ and the x exclusive segment in $p_6 = (4, 2.5)$.

The variable gadget is shown in Figure 3.2.

Lemma 3.7. *Exactly one of the variable literal x segment and the variable literal $\bar{x}$ segment has the color true, and the other one has the color false.*

Proof. All segments in this gadget except the variable neutral segment intersect with the variable neutral segment. Due to this, none of these four segments may have the neutral color, and limits their choice to the two colors true and false. Let a be the color of the x exclusive segment. This segment intersects with both the $\bar{x}$ exclusive segment and the $\bar{x}$ variable literal segment, so they both have the same color b with $b \neq a$. The x variable literal segment intersects with the $\bar{x}$ exclusive segment, so it has the color a. □

This only shows limitations for the coloring of these two segments, but a gadget that only allows for the x variable literal segment to have the color true would also fulfill Lemma 3.7. This motivates the upcoming Lemma 3.8, which proves that when only considering the restrictions of the gadet, there exists a coloring that assigns the x variable literal segment the color true and one that assigns this segment the color false.

Lemma 3.8. *There exist at least two colorings of this gadget, one where the variable literal x segment has the color true, and one where the variable literal x segment has the color false.*

Proof. The proof of Lemma 3.7 makes two restrictions on the colors a, b: They may not be the neutral color, and they may not be equal. The proof makes use of all intersections in the gadget, which means that there are no further restrictions. This means that b, the color of the variable literal x segment, may be the color true or false and both result in a proper coloring of this gadget. □

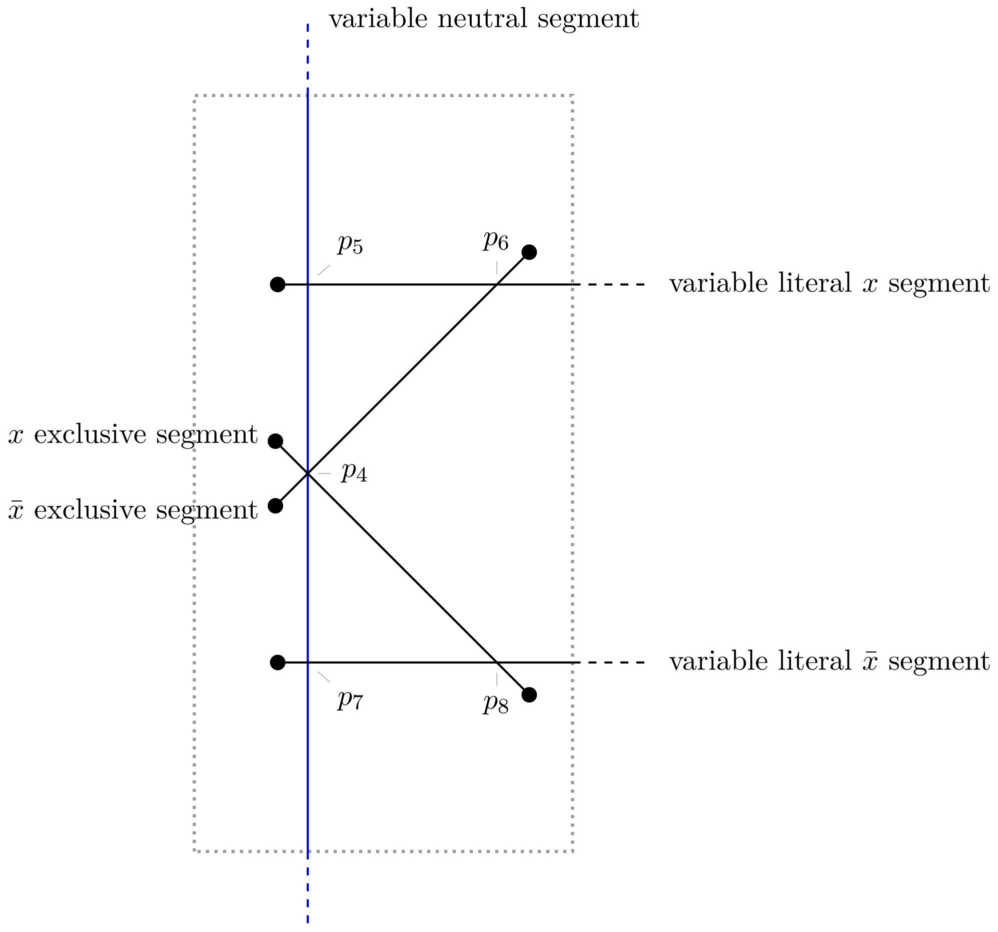
variable neutral segment
p_5
p_6
variable literal x segment
x exclusive segment
p_4
$\bar{x}$ exclusive segment
variable literal $\bar{x}$ segment
p_7
p_8

Figure 3.2.: The variable gadget. All fixed endpoints are marked with dots. A dashed line at the end of a segment indicates that there is no fixed end to it, and it may be extended.

- A *clause gadget.* This gadget is placed once for every clause $c \in C$ in the 3-SAT instance. Let x, y, z be the literals of the clause c. Then, the gadget representing c in the final construction contains three segments, each of them with the same color as the variable literal x, y, z segment, respectively. The entire gadget is constructed in such a way that is only colorable if at least one of these segments has the color true. This gadget may use the clause segments from the base gadget, but they must be able to be extended through the gadget to be available for other clause gadgets to the right of the current one.

  The bounding box of the clause gadget spans from $(0, 0)$ to $(10, 15)$. It contains the following eleven segments:

  - The clause false segment, which is already defined in the base gadget, and the clause neutral segment, which is already defined in the base gadget. They pass through every instance of this gadget, from $(0, 1.5)$ to $(15, 1.5)$ and $(0, 2.5)$ to $(15, 2.5)$, respectively.
  - The *clause c result segment* with endpoints $(10, 1)$ and $(10, 4.5)$. It has slope $\pi/2$. This segment intersects the clause false segment in the point $p_9 = (10, 1.5)$ and the clause neutral segment in the point $p_{10} = (10, 2.5)$.
  - The *clause c intermediate result segment* with endpoints $(5, 2)$ and $(5, 5.5)$. It has slope $\pi/2$. This segments intersect the clause neutral segment in the point $p_{11} = (5, 2.5)$.

- The *intermediate result bridge segment* with endpoints $(4.5, 4)$ and $(10.5, 4)$. It has slope 0. This segment intersects the clause c result segment and the clause c intermediate result segment in the points $p_{12} = (10, 4)$ and $p_{13} = (5, 4)$, respectively. The intermediate result bridge segment is below the clause neutral segment.
- The *clause c literal x bridge segment* with endpoints $(5.5, 4.5)$ and $(2, 8)$, and *clause c literal y bridge segment* with endpoints $(4.5, 4.5)$ and $(8, 8)$, These segments have the slope $3\pi/4$ and $\pi/4$, respectively. These segments intersect each other and the intermediate result segment in the point $p_{14} = (5, 5)$, which is below the intermediate result bridge segment.
- The *clause c literal x segment* with endpoints $(2.5, 7)$ and $(2.5, y_{\text{end}c,x})$, $y_{\text{end}c,x} > 10$, and *clause c literal y segment*, with endpoints $(7.5, 7)$ and $(7.5, y_{\text{end}c,y})$, $y_{\text{end}c,y} > 10$. Both of these segments have slope $\pi/2$. These two segments intersect clause c literal x bridge segment in $p_{15} = (2.5, 7.5)$ and clause c literal y bridge segment in $p_{16} = (7.5, 7.5)$, respectively.
- The *clause c literal z bridge segment* with endpoints $(9.5, 3.5)$ and $(13, 7)$. It has slope $\pi/4$. This segment intersects the clause c result and the intermediate result bridge segment in the point p_{12}.
- The *clause c literal z segment* with endpoints $(12.5, 6)$ and $(2.5, y_{\text{end}c,z})$, $y_{\text{end}c,z} > 10$ It has slope $\pi/2$. It intersects the lause c literal z bridge segment in the point $p_{17} = (12.5, 6.5)$.

The clause gadget is shown in Figure 3.3.

Lemma 3.9. *The clause gadget for the clause c is not colorable if all three clause c literal segments have the color false.*

Proof. If the clause c literal x and literal y segments have the color false, the clause c literal x and literal y bridge segments both may not have the color false. Because they may not have the same color, one of them has the color true and the other one neutral. The intermediate result segment, which intersects both of these, must therefore have the color false.

If the intermediate result segment and the literal z segments have the color false, the clause c literal z and intermediate result bridge segment both may not have the color false. Because they may not have the same color, one of them has the color true and the other one neutral. The result segment, which intersects both of these, must therefore have the color false. This is a contradiction to the fact that the result segment may not have the color false due to its intersection with the clause false segment.

Therefore, a proper coloring of this gadget does not exist if all three clause c literal segments have the color false. ☐

Lemma 3.10. *If of the clause c variable x, y, z segments, none has the neutral color and at least one has the true color, then the gadget is colorable.*

Proof. The Table 3.1 shows one possbile coloring for each of the seven colorings of the clause literal segments that fulfill the condition of Lemma 3.10. The colors true, false and neutral are shortened to t, f and n, respectively.

Segment name	c_1	c_2	c_3	c_4	c_5	c_6	c_7
clause c literal x segment	t	f	t	f	t	f	t
clause c literal y segment	f	t	t	f	f	t	t
clause c literal z segment	f	f	f	t	t	t	t
clause c literal x bridge segment	f	n	f	n	f	n	f
clause c literal y bridge segment	n	f	n	t	n	f	n
clause c literal z bridge segment	n	n	n	f	n	n	n
intermediate result segment	t	t	t	f	t	t	t
intermediate result bridge segment	f	f	f	n	f	f	f
result segment	t	t	t	t	t	t	t

Table 3.1.: Possible colorings for the clause gadget under the conditions outlined in Lemma 3.10.

□

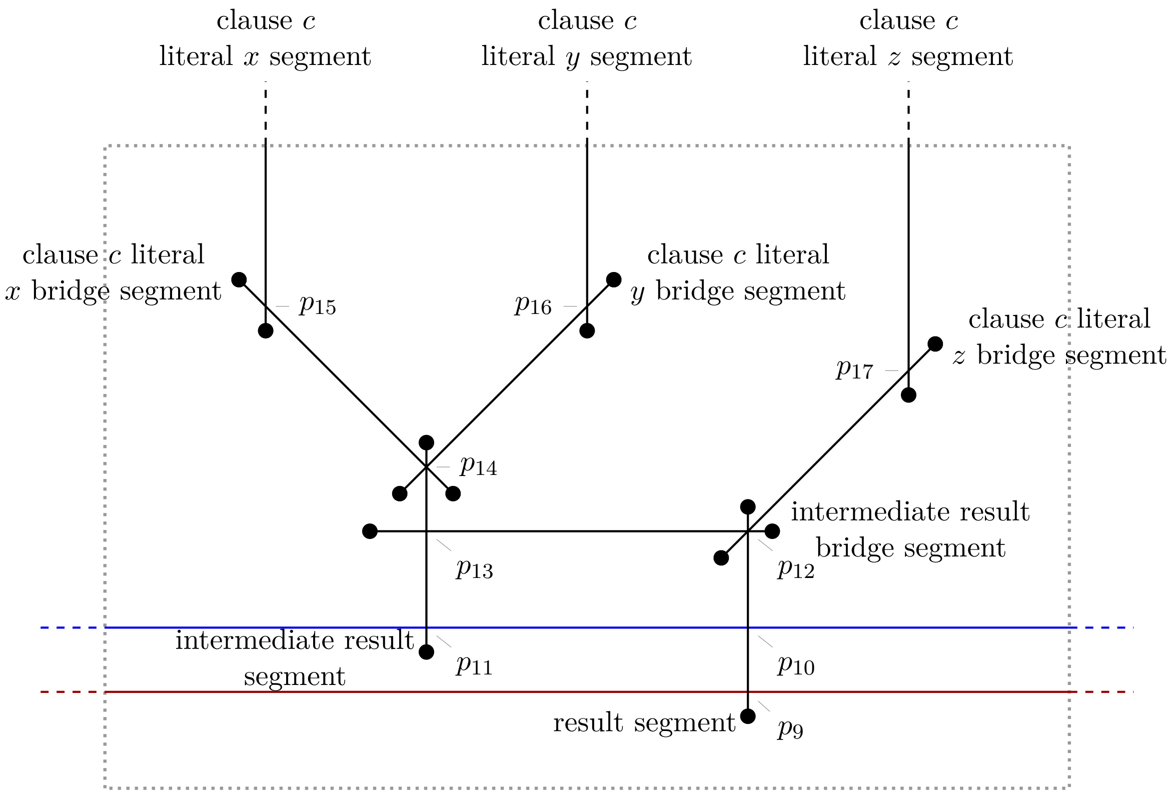
clause c literal x segment
clause c literal y segment
clause c literal z segment
clause c literal x bridge segment
p_{15}
p_{16}
clause c literal y bridge segment
clause c literal z bridge segment
p_{17}
p_{14}
intermediate result bridge segment
p_{13}
p_{12}
intermediate result segment
p_{11}
p_{10}
result segment
p_9

Figure 3.3.: The clause gadget. All fixed endpoints are marked with dots. A dashed line at the end of a segment indicates that there is no fixed end to it, and it may be extended.

- A *distribution gadget.* After assembling the base, variable and clause gadgets, the segments colored with the value of their literal need their need to be directed into the clause gadgets. To do this, we create a gadget that can be attached to the variable literal segments and provides a segment with the same color that can be used by the clause gadgets.

  The bounding box of the distribution gadget spans from $(0, 0)$ to $(5, 5)$. It contains the following four segments:

- The variable literal x segment for some literal x, which is already defined in the variable gadget. A segment with some literal x passes through every instance of this gadget, from $(0, 2.5)$ to $(5, 2.5)$.
- A *distribution left segment* with endpoints $(1, 3)$ and $(3, 1)$. It has slope $\pi/4$. It intersects the variable literal x segment in $p_{18} = (1.5, 2.5)$.
- A *distribution right segment* with endpoints $(2, 1)$ and $(4, 3)$. It has slope $3\pi/4$, It intersects the variable literal x segment in $p_{19} = (3.5, 2.5)$ and the distribution left segment in $p_{20} = (2.5, 1.5)$.
- A clause c literal x segment for the literal x for which the variable literal x segment is also in this gadget. The clause c literal x segment has endpoints $(2.5, 2)$ and $(2.5, y), y < 0$. This segment intersects the distribution left and right segments in p_{20}.

Lemma 3.11. *The variable literal x segment and the clause c literal x segment have the same color in a proper 3-coloring.*

Proof. Let a be the color of the variable literal x segment in a proper 3-coloring. The left distribution segment may not have the color a because it intersects the variable literal x, let its color be $b \neq a$. The right distribution segment may not have the color a because it intersects the variable literal x, and it may not have the color b because it intersects the left distribution segment, let its color be c with $c \neq a$ and $c \neq b$.

The clause c literal x segment intersects both the left and right distribution segments, therefore it may not have the color b or c. The only remaining option for this segment is color a. □

The distribution gadget is shown in Figure 3.4. The two segments marked in orange need to have the same color in a proper 3-coloring.

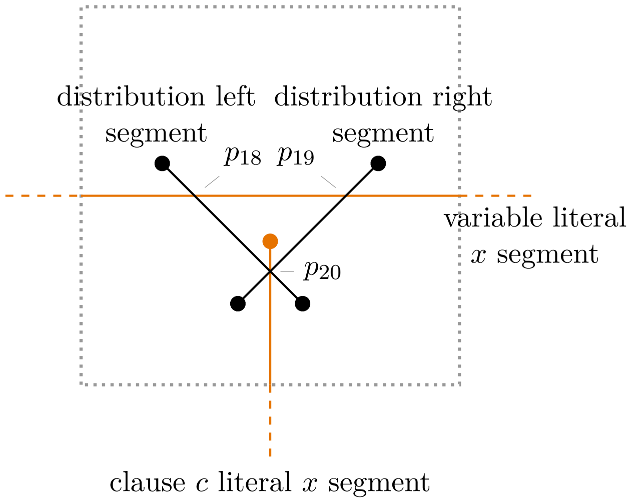
distribution left segment
distribution right segment
p_{18}
p_{19}
variable literal x segment
p_{20}
clause c literal x segment

Figure 3.4.: The distribution gadget. All fixed endpoints are marked with dots. A dashed line at the end of a segment indicates that there is no fixed end to it, and it may be extended.

- **A crossing gadget.** In most constructions, a clause literal segment needs to pass over a variable literal segment, even though they have the same color. Because this is not supposed to invalidate the coloring, we need a gadget that allows segment with the same and differing colors to cross. This means that the segments may be split, but both parts of that split must have the same color, which is enforced by the gadget. To simplify the upcoming construction, both parts of the split segment have the same name.

The bounding box of the clause gadget spans from $(0, 0)$ to $(5, 5)$. It contains the following 14 segments:

- The segment a, which is already defined by another gadget, with endpoints $(x_{\text{a-left}}, 2.5)$, $x_{\text{a-left}} < 0$ and $(0.7, 2.5)$ for its left part, and endpoints $(2.3, 2.5)$ and $(x_{\text{a-right}}, 2.5)$, $x_{\text{a-right}}$ for its right part.
- The segment b, which is already defined by another gadget, with endpoints $(2.5, y_{\text{b-lower}})$, $x_{\text{b-lower}} < 0$ and $(2.5, 2.2)$ for its lower part, and endpoints $(2.5, 3.5)$ and $(2.5, y_{\text{b-upper}})$, $x_{\text{a-right}}$ for its upper part.
- The *horizontal cross 1 segment*, with endpoints $(0.3, 2.3)$ and $(1.1, 3.1)$. It has slope $\pi/4$. It intersects the segment a in $p_{21} = (0.5, 2.5)$.
- The *horizontal cross 2 segment*, with endpoints $(0.3, 2.7)$ and $(1.1, 1.9)$. It has slope $3\pi/4$. It intersects the segment a and the horizontal cross 1 segment in p_{21}.
- The *horizontal cross 3 segment*, with endpoints $(0.9, 0.7)$ and $(0.9, 4.3)$. It has slope $\pi/2$. It intersects the horizontal cross 1 segment in $p_{22} = (0.9, 2.9)$ and the horizontal cross 2 segment in $p_{23} = (0.9, 2.1)$.
- The *horizontal cross 4 segment*, with endpoints $(0.7, 4.3)$ and $(2.7, 2.3)$. It has slope $3\pi/4$. It intersects the horizontal cross 3 segment in $p_{24} = (0.9, 4.3)$.
- The *horizontal cross 5 segment*, with endpoints $(0.7, 4.3)$ and $(2.7, 2.3)$. It has slope $\pi/4$. It intersects the horizontal cross 3 segment in $p_{25} = (0.9, 0.7)$ and the horizontal cross 4 segment and segment a in $p_{26} = (2.5, 2.5)$.
- The *vertical cross 1 segment*, with endpoints $(1.5, 3)$ and $(2.7, 1.8)$. It has slope $3\pi/4$. It intersects segment b in $p_{27} = (2.5, 2)$ and the horizontal cross 5 segment in $p_{28} = (2.25, 2.25)$.
- The *vertical cross 2 segment*, with endpoints $(1.5, 3)$ and $(2.7, 1.8)$. It has slope $3\pi/4$. It intersects segment b and the vertical cross 1 segment in p_{27} and segment a in $p_{29} = (3, 2.5)$.
- The *vertical cross 3 segment*, with endpoints $(1, 2.8)$ and $(4, 2.8)$. It has slope 0. It intersects the horizontal cross 4 segment in $p_{30} = (2.2, 2.8)$, the vertical cross 1 segment in $p_{31} = (3.3, 2.8)$, and the vertical cross 2 segment in $p_{32} = (1.7, 2.8)$.
- The *vertical cross 5 segment*, with endpoints $(4, 2.6)$ and $(2.2, 4.4)$. It has slope $3\pi/4$. It intersects segment b in p_{33} and the vertical cross 3 segment in $p_{36} = (3.8, 2.8)$.

The crossing gadget is shown in Figure 3.5. Both segments in orange and both segments in teal need to have the same color, respectively. The dotted orange and teal lines are not segments and only serve to visualize the alignment requirements.

Lemma 3.12. *Both parts of segment a have the same color, and both parts of segment b have the same color.*

Proof. Let a be the color of the left part of segment a. Because the horizontal cross 1 and horizontal cross 2 segment intersect each other and the left part of segment a, the have the colors b and c, where a, b, c are pairwise different.

Because the horizontal cross 3 segment intersects both the horizontal cross 1 and horizontal cross 2 segment, it must have color a. Because the horizontal cross 4 and horizontal cross 5 segment intersect each other and the horizontal cross 3 segment, the have the colors b and c.

The right part of segment a intersects both the horizontal cross 4 and horizontal cross 5 segment, and must therefore have the color a.

Analogue for segment b and the vertical cross segments. □

Lemma 3.13. *There is at least one coloring for each combination of colors for segment a and b.*

Proof. The Table 3.2 shows two possilbe colorings, one where segment a and b have the same color, and one where they have different colors. A coloring for every other combination of colors for segment a and b than presented can be acquired by permuting one of these two colorings. The colors true, false and neutral are shortened to t, f and n, respectively.

Segment name	c_1	c_2
segment a	t	t
segment b	t	f
horizontal cross 1 segment	f	f
horizontal cross 2 segment	n	n
horizontal cross 3 segment	t	t
horizontal cross 4 segment	n	n
horizontal cross 5 segment	f	f
vertical cross 1 segment	n	t
vertical cross 2 segment	f	n
vertical cross 3 segment	t	f
vertical cross 4 segment	f	t
vertical cross 5 segment	n	n

Table 3.2.: Possible colorings for the cross gadget.

□

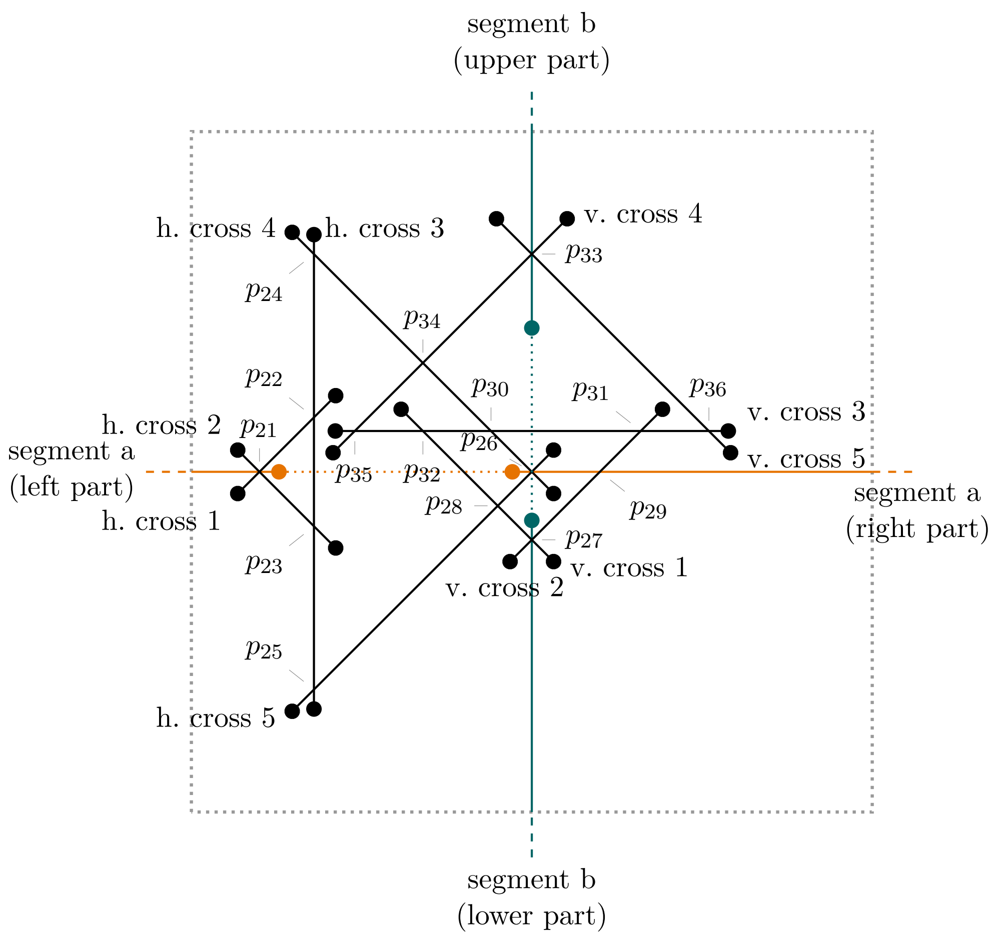

Figure 3.5.: The crossing gadget. All fixed endpoints are marked with dots. A dashed line at the end of a segment indicates that there is no fixed end to it, and it may be extended.
The labels of the horizontal/vertical cross x segments are abbreviated with h./v. cross x.

We now construct the representation R of a graph that is 3-colorable if and only if the original 3-SAT (U, C) instance has a boolean assignment $t : U \to \mathbb{B}$ that satisfies all clauses in C.

1. Place a single base gadget at $(0, 0)$. This gadget marks the bottom left corner of the final representation. There are two variables defined in this gadgets description. We choose $x_{\text{end}} = 6 + |C| \cdot 15$ and $y_{\text{end}} = 11 + |U| \cdot 10$.
2. For each variable $x_i \in U$ in the 3-SAT instance, place a variable gadget at $(0, 10 + (i - 1) \cdot 10)$. The upper border of the last of these gadgets is at $h = 10 + |U| \cdot 10$. Because $h < y_{\text{end}}$, the variable neutral segment extend through all variable gadgets placed.
3. For each clause $c_i \in C$ in the 3-SAT instance, place a clause gadget at $(5+(i-1)\cdot 15, 0)$. The right border of the last of these gadgets is at $w = 5 + |C| \cdot 15$. Because $w < x_{\text{end}}$, the clause false and neutral segment extend through all clause gadgets placed.
4. Extend the variable literal segments from the variable gadgets as far right as the clause segments from the base gadget.
5. For each clause literal v segment with x-position x_v, let y_v be the y-position of the variable literal v segment. Set the upper endpoint of the clause literal v segment to

$(x_v, y_v - 0.5)$ and place a distribution gadget at $(x_v - 2.5, y_v - 2.5)$, connecting the two literal segments.

6. After placing all distribution gadget in step 5, place one crossing gadget at $(x_{\text{cross}} - 2.5, y_{\text{cross}} - 2.5)$ for each intersection between a variable literal x and a clause c literal y segment at $(x_{\text{cross}}, y_{\text{cross}})$.

An illustration of a complete construction is in Figure 3.6.

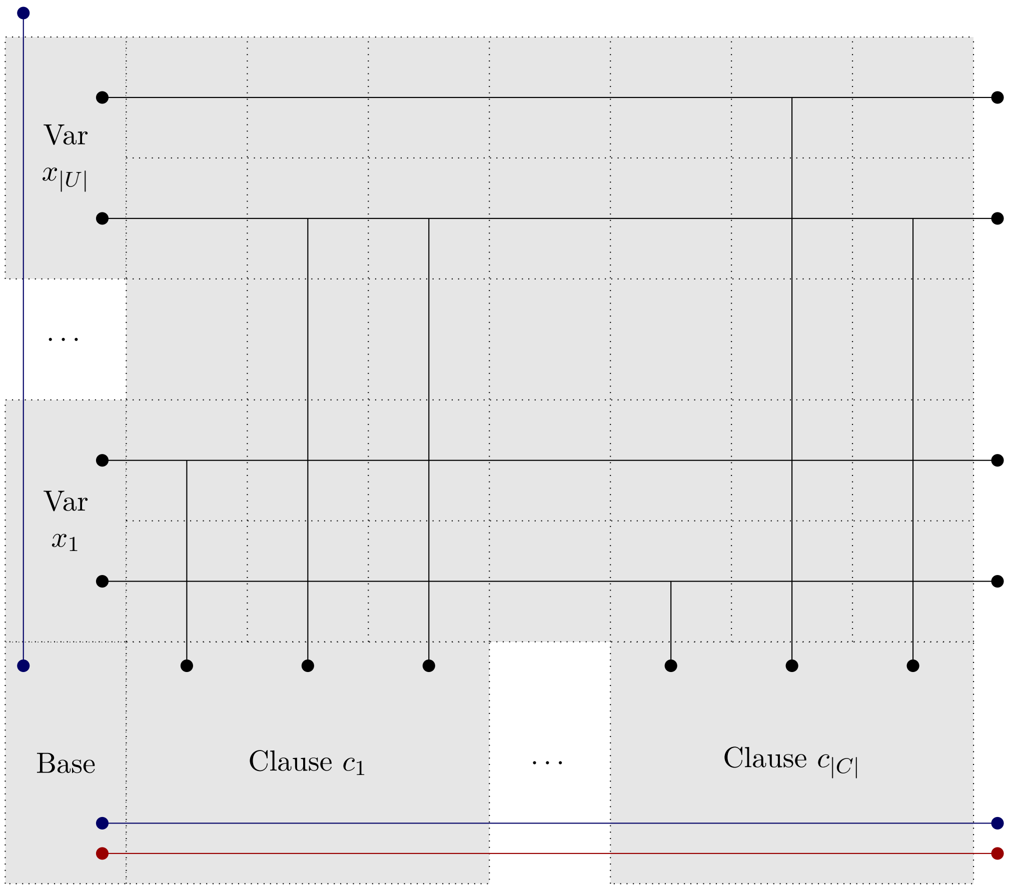

Figure 3.6.: The general structure of a completed 3-colorability construction.

Proof of correctness

In this section, we show that the construction R above requires polynomial time and is 3-colorable if and only if the original 3-SAT instance (U, C) has a satisfying boolean assignment t.

First, we show that from a valid 3-coloring of this graph, one can construct a valid boolean assignment for the variables U.

For each variable $x \in U$, assign it the value represented by the color of the variable literal x segment in the boolean value assignment $t : U \to \mathbb{B}$. Because of Lemma 3.7, which states that the color of this segment can not be the neutral color, this is either true or false.

Let $c \in C$ be a clause and x a literal in that clause. From the construction and Lemma 3.11, we know that the variable literal x segment and the clause c literal x segment have the same color. Let x, y, z be the literals of the clause c. From Lemma 3.9, we know that in a valid coloring, the variable literal segment of x, y or z has to be true, which means that $t(x), t(y)$ or $t(z)$ is true, satisfying the clause c. Because we made no restrictions on the clause $c \in C$, every clause is satisfied by t.

Next, we show that from a satisfying boolean assignment t for the variables in U, one can construct a proper coloring of this graph.

First, choose which color represents true, false and neutral, and color the base gadget as described in Lemma 3.6. Then, color the variable literal segments with the color representing the value of this literal in the boolean assignment. This satisfies the condition of Lemma 3.7 that one of these segments has the color false and the other the color true, and is possible because of Lemma 3.8.

From this initial coloring, the distribution gadgets are colorable by Lemma 3.11 and the crossing gadgets are colorable by Lemma 3.13. Let $c \in C$ be a clause with the literals x, y, z. Because we colored the variable literal segments with the assignments of t, at least one of the clause c literal x, y, z segments has the color true following Lemma 3.11 for the distribution gadgets and Lemma 3.12 for the crossing gadgets. From Lemma 3.10 follows that the gadget for clause c is colorable. Because we made no restrictions on the clause $c \in C$, every clause gadget is colorable.

Last, we show that this can construction can be built in polynomial time. This construction requires $\mathcal{O}(|U| \cdot |C|)$ segments. Each gadget only requires a constant amount of segments. Each segment in the final construction is accounted for at least once by a gadget. In total, there is one base gadget, $|U|$ variable gadgets, and $|C|$ clause gadgets. A distribution or crossing gadget is only placed where a clause literal segment and a variable literal segment intersect. There are at most $2|U| \cdot 3|C|$ such intersections and therefore at most this many distribution and crossing gadgets.

NP-hardness of Pure-4-Dir(d_1, d_2, d_3, d_4)-3-Color

After establishing that the coloring problem is NP-hard for representations with four fixed slopes, we show a generalization of the proof that allows for any set of four slopes in the representation.

Theorem 3.14. *The problem* PURE-4-DIR(d_1, d_2, d_3, d_4)-3-COLOR *is NP-hard for any set* $\{d_1, d_2, d_3, d_4\}$ *with* $d_1, d_2, d_3, d_4 \in [0, \pi/2)$.

To simplify this proof, we first show that we may restrict the representations without limiting the graph classes we examine. Quapil proved in [Qua21] that for any set of four slopes, $\{d_1, d_2, d_3, d_4\}$ with $d_1 < d_2 < d_3 < d_4$, an invertible linear transformation f exists which fulfills $f(d_1) = \pi/4, f(d_2) = \pi/2, \pi/2 < f(d_3) < 3\pi/4, f(d_4) = 3\pi/4$. We outline a similar proof that shows that there is an invertible linear transformation f with $f(d_1) = 0, f(d_2) = \pi/4, \pi/4 < f(d_3) < 3\pi/4, f(d_4) = 3\pi/4$.

1. Scale along the horizontal axis by a positive scalar such that $d_4 - d_2 = \pi/2$.
2. Rotate such that $d_2 = \pi/4$. After this rotation, $d_4 = 3\pi/4$ is also set.
3. Scale along the axis with slope $\pi/4$ by a positive scalar such that $d_1 = 0$. This scaling leaves the slopes d_2 and d_4 unmodified.

None of these operations change the relative order of the slopes, which implies that $d_2 < d_3 < d_4$ is still true, and therefore $\pi/4 < d_3 < 3\pi/4$ is fulfilled. It follows that the graph class 4-Dir

$$\text{4-Dir} = \bigcup_{d_3 \in (\pi/4, 3\pi/4)} \text{4-Dir}(0, \frac{\pi}{4}, d_3, \frac{3\pi}{4}) \tag{3.1}$$

All the gadgets described in section 3.1 are still representable with the slopes $\{0, \pi/4, d_3, 3\pi/4\}$ with $\pi/4 < d_3 < 3\pi/4$. The remaining proof is equivalent to the proof for four fixed slopes. The construction of figure 3.6 is skewed such that the vertical segments have slope d_3 instead. Because the resulting intersection graph is exactly the same, the correctness and limited size of the graph still hold.

Tightness of the result

The use of four directions is the minimum possible. PURE-3-DIR-3-COLOR, like any PURE-K-DIR-K-COLOR, is colorable due to the possibility of mapping each direction to a color, and giving each segment a color according to its direction. Two segments of the same direction never intersect in Pure-k-dir graphs, so this is a valid coloring.

3.2. NP-completeness of Pure-$(k+1)$-Dir-k-Color

Theorem 3.15. *For each $k \in \mathbb{N}$ with $k \geq 3$, the problem* PURE-$(k+1)$-DIR-k-COLOR *is NP-complete.*

We already proved this for $k = 3$ in the last subsection. Assume that for any fixed $k \in \mathbb{N}$ with $k \geq 3$, the problem PURE-$(k+1)$-DIR-k-COLOR is NP-complete. Let $R = (P, S)$ be a representation of a graph in Pure-$(k+1)$-Dir. Examine the problem for $k' = k + 1$. The goal is to create a graph in Pure-$(k'+1)$-Dir which is colorable exactly if the graph represented by R is colored. The representation we construct is denoted by R'. The representations of graphs in Pure-$(k'+1)$-Dir may use one more slope than the graphs in Pure-$(k+1)$-Dir, denoted $d_{k'+1}$. We build R' from R:

1. As long as there is a segment $s \in S$ which does not have the slope $d_{k'+1}$ and is not intersected by one with slope $d_{k'+1}$, add a segment s' with slope $d_{k'+1}$ that intersects s. Extend s' until no more intersections can be created by extending it further.

   Of these new segments, no two intersect. If s_1 and s_2 were intersecting, they would also both intersect the segment s for which s_2 was placed. Because s_1 already intersects s, the new segment s_2 would not have been placed.

2. Create a clique of order k by adding k segments with exactly one of them with slope $d_1, \ldots, d_k$ that all intersect in a single point p. The choice of p and the length of these k segments has to be in a manner that no other intersection are created, and the next step is still feasible.

3. Extend each segment with slope $d_{k'+1}$ until it intersects all segments created in the previous point.

The segments created in step 2 all have the same different colors, so all segment created in step 1 have the same color. Therefore, the original graph may not use this color. It follows that, for each $k, l \in \mathbb{N}$ with $l > k \geq 3$, the problem PURE-l-DIR-k-COLOR is NP-hard.

Corollary 3.16. *For each $k \in \mathbb{N}$ with $k \geq 3$, the problem $(k+1)$-*DIR*-k-*COLOR *is NP-complete.*

Proof. This follows from Lemma 3.4, Pure-k-Dir $\subseteq$ k-Dir for all k, and Theorem 3.15. □

3.3. NP-completeness of Seg-k-COLOR

This section concludes this chapter with the proof for Theorem 3.2, which we set out to prove in this chapter.

Proof of Theorem 3.2. The problem PURE-$(k+1)$-DIR-k-COLOR is NP-complete, see Theorem 3.15. Further, Pure-$(k+1)$-Dir is a subclass of Seg. From this and Lemma 3.4 follows that SEG-k-COLOR is NP-hard.

The proof that SEG-k-COLOR is in NP is Lemma 3.3. □

4. Instance analysis

This chapter covers the analysis of the 225 competition instances. We show general statistical properties for these instances and further properties that are of interest for specific instance types. While coloring methods are used to motivate the examination of these properties, the coloring methods are not introduced and evaluated in detail here, but in Chapter 5.

Because of Lemma 2.2, we are especially interested in properties that are invariant under invertible linear transformations and translations. This includes all properties of the intersection graphs themselves.

All competition instances belong to one of five types: reecn, sqrp, sqrpecn, visp, or vispecn. Some competition instances have an extra r in front of their name, for example rsqrp7320, but these are generated in the exact same way as their counterparts without an r. The exact generation methods are detailed in their respective analysis section.

4.1. General analysis

Here, we present statistics that are relevant for all instance types. Note that some of these statistics are biased towards instance types with larger instances. Table 4.1 provides an overview over the number of instances and the sum of segments and intersections grouped by instance type.

Instance type	Count	Total segments	Total intersections
reecn	45	1 702 324	9 611 195 542
sqrp	45	1 781 370	22 586 027 646
sqrpecn	45	1 677 212	18 854 770 863
visp	45	1 657 503	4 016 500 891
vispecn	45	1 528 924	3 409 252 108
Total	225	8 357 333	58 477 747 050

Table 4.1.: Number of instances and sum of size for each instance grouped by instance type.

4.1.1. Representations

The representations are graphs themselves. In this section, we analyze the graph structure and geometry of the representations.

Vertex covers of endpoints

Two segments with a common endpoints rarely intersect, mainly because this requires a set of three collinear points. While collinear points are not impossible, they are very rare due to their random generation on an integer grid with large bounds. This motivates the search for vertex covers of endpoints in the representation.

We use the heuristics presented in [ACL12] to find a vertex cover. All of them start with an empty, incomplete vertex cover $C = \emptyset$ and add vertices to it until it covers all edges. All vertices V are visited in some order $\sigma : \mathbb{Z}_{|V|} \to V$. After a vertex v was visited, all incident edges $\{uv \in E \mid u \in V\}$ are covered. Some strategies use the *label* $L(v) := \sigma^{-1}(v)$ of a vertex v when deciding which vertices to add to the cover.

- **LR:** The strategy ListRight is defined in [DL08]. If the currently visited vertex u is not in the cover, add all of its neighbors that are not in the cover C to it. This means adding all vertices v with $\{v \mid uv \in E \wedge v \notin C\}$ to C.
- **ED:** This strategy is an approximation algorithm that requires at most twice as many vertices as an optimal cover [ACL12]. If a vertex $u \notin C$ has a neighbor $v \notin C$, then both u and v are added to C.
- **S-Pitt:** This is a variation of the ED algorithm. If a vertex $u \notin C$ has a neighbor $v \notin C$, then u or v is added to C. Which one is chosen uniformly at random.
- **LL:** The strategy ListLeft is defined in [ACL11]. The vertex u is added to the cover C if there is a neighbor v that has a greater label than the current vertex $L(v) > L(u)$.
- **SLL:** The strategy Sorted-ListLeft is defined in [ACL11]. The vertex u is added to the cover C if it has a neighbor v that has a lower degree $d(v) < d(u)$. If the degrees of these two vertices are equal $d(v) = d(u)$, the vertex is added under the condition of LL, which is $L(v) > L(u)$.
- **ASLL:** The strategy Anti Sorted-ListLeft is defined in [ACL11]. The vertex u is added to the cover C is there is a neighbor v that has a lower degree $d(v) > d(u)$. If the degrees of these two vertices are equal $d(v) = d(u)$, the vertex is added under the inverted condition of LL, which is $L(v) < L(u)$.

Additionally, we introduce the following strategies:

- **GD:** This algorithm iterates over all edges $uv \in E$. If one endpoint is already in the cover, formally $u \in C \vee v \in C$, no new vertex is added to the cover C. Otherwise, the vertex with the greater degree is added. The motivation for this heuristic are the sqrp instances, where segments are more likely to have one of its endpoints close to the border, giving these endpoints high degrees.
- **inward:** The center of a representation is the average over all points of the representation. Order the vertices non-increasingly by their distance from the center, then use the ListLeft strategy. The motivation for this heuristic is the same as the one for GD.
- **upward:** Order the vertices non-decreasingly by their height, then use the ListRight strategy. This strategy is motivated by the intuition of creating 'stripes', intervals of y-positions, that alternate between taking the points in that interval into the cover and not doing so.

Evaluation of vertex covers

Table 4.2 shows how often each heuristic outlined above gives the best result grouped by instance type. Heuristics that do not give the best result on any instance are omitted. If two heuristics give the best result, both of them are counted. Because this happens ten times on the competition instances, the total number of the best solutions adds up to 235.

Instance type	LR	S-Pitt	SLL	GD	upward
reecn	22	0	0	0	23
sqrp	6	0	5	37	3
sqrpecn	29	0	0	0	18
visp	0	6	0	6	35
vispecn	6	0	0	0	39
Total	63	6	5	43	118

Table 4.2.: The number of times each heuristic gives the best result for an instance grouped by instance type.

Slopes

Each segment of a geometric graph corresponds to one vertex of the intersection graph. Because the competition instances are randomly generated, we analyse the distribution of slopes across these instances.

Figure 4.1 shows the portion of segments up to a given slope compared to the cumulative density function of a uniform distribution. We use cumulative instead of probability functions because competition instances provide a discrete distribution in contrast to the continuous distribution we compare against.

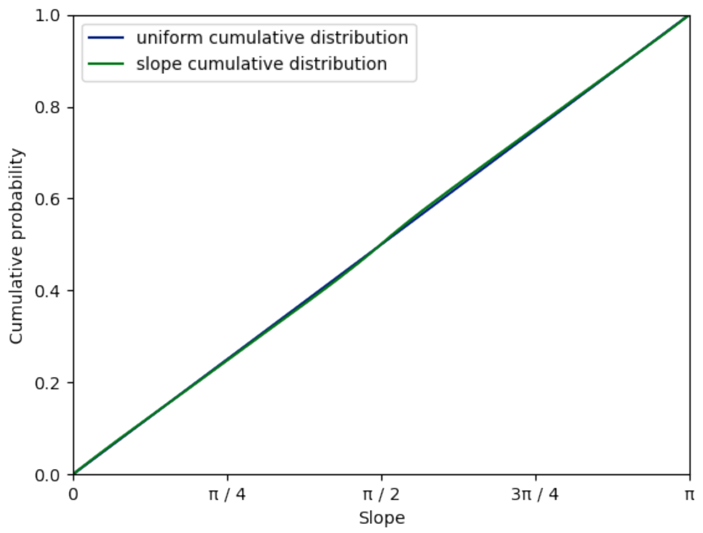

Figure 4.1.: The cumulative distribution of the slopes over all instances and the uniform distribution $U(0, \pi/2)$.

The similarity between these two shows that a random variable following a uniform distribution $S \sim U(0, \pi/2)$ is a good model for the slopes of segments. Following this observation, let $S_1, S_2 \sim U(0, \pi/2)$ be the slope of two intersecting segments. The random variable X_{12} describing the intersection angle of these two segments has the distribution $\min(|S_1 - S_2|, \pi - |S_1 - S_2|)$, which is equal to the uniform distribution $U(0, \pi/4)$.

4.1.2. Intersection graph

The graph coloring problems has been studied extensively, and for some graph classes, efficiently coloring algorithms were developed. The goal when analyzing the intersection graphs is to find out whether they or large subgraphs of them are in these graph classes.

Clique sizes

Let $\omega(G)$ be the size of the largest clique of the graph G. This is a lower bound on the chromatic number of the graph $\chi(G) \geq \omega(G)$.

Because calculating the size of the largest clique is NP-complete, we use Matula's coloring algorithm [MMI72] as a heuristic to find large cliques in the intersection graphs. In this section, we only introduce its use in finding cliques, not how it is used to color graphs.

The idea of this algorithm for an input graph G is to remove the vertex with the lowest degree and its incident edges until only a complete graph remains. This graph is a subgraph of G and complete, therefore it is a clique of G.

Algorithm 4.1: Matula clique

```
   Input: Graph $G = (V, E)$
   Output: Size of a clique $\omega \in \mathbb{N}$
1  while $\min(\{d(v) \mid v \in V\}) + 1 < |V|$ do
2      $v \in V$ is a vertex with minimal degree $d(v)$;
3      $G \leftarrow G - v$;
4  return $|V|$
```

Algorithm 1 shows a formal description of this algorithm. Next, we show that this algorithm always outputs a clique. Let $V^{(i)}, E^{(i)}$ be the sets of vertices and edges after i iterations of the while loop. The input graph is $(V^{(0)}, E^{(0)})$. In each iteration, elements are removed from V and E, but no new ones are added. Let k be the number of iterations the while loop makes. The resulting graph after the while loop $(V^{(k)}, E^{(k)})$ is a subgraph of the input graph $(V^{(0)}, E^{(0)})$. This can formally be described as follows:

$$\begin{aligned} \forall i \in \mathbb{Z}_k : V^{(i)} &\subseteq V^{(i-1)} \\ \implies V^{(k)} &\subseteq V^{(0)} \\ \forall i \in \mathbb{Z}_k : E^{(i)} &\subseteq E^{(i-1)} \\ \implies E^{(k)} &\subseteq E^{(0)} \end{aligned}$$

Therefore, $(V^{(i)}, E^{(i)})$ is a subgraph of the input graph for all i. Further, we know that the minimum degree of $(V^{(k)}, E^{(k)})$ is $|V^{(k)}| - 1$, as the loop terminates. The maximum degree is also $|V^{(k)}| - 1$, which means that every vertex has degree $|V^{(k)}| - 1$ and the graph is a complete graph.

This algorithm runs in $\mathcal{O}((|V| + |E|)\log(|V|))$ with a data structure that perform the following operations in $\mathcal{O}(\log(V))$ or faster: add a vertex, reduce the degree of a vertex by one, find the index of a vertex with lowest degree, remove a vertex, and get the minimal degree of all vertices. A segment tree accomplishes this [BW80].

Figure 4.2 shows the clique sizes found with Matula's algorithm compared to the number of vertices in the intersection graph.

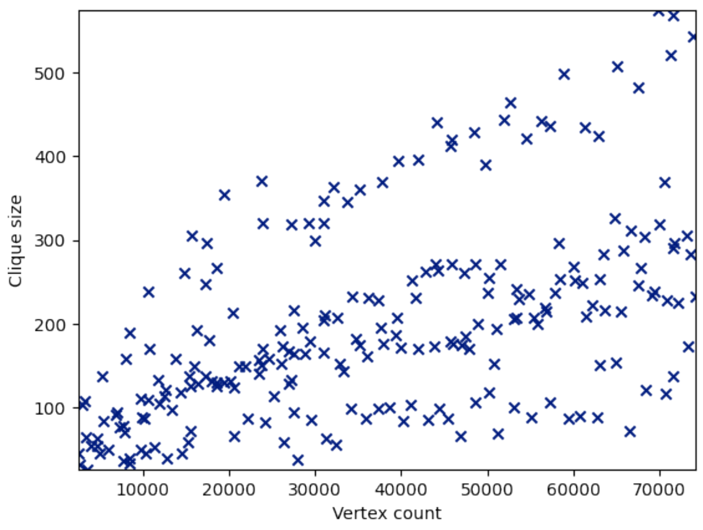

Figure 4.2.: The size of cliques plotted against the number of segments. Each marker represents one instance.

Diameter

In our analysis of the competition instances, we calculated the diameters of some instances. While this is not directly motivated by a coloring method, it is still of interest for the study of this random graph model.

Figure 4.2 shows the diameters compared to the number of vertices for some intersection graph. This diagram does not show the diameters for all intersection graphs.

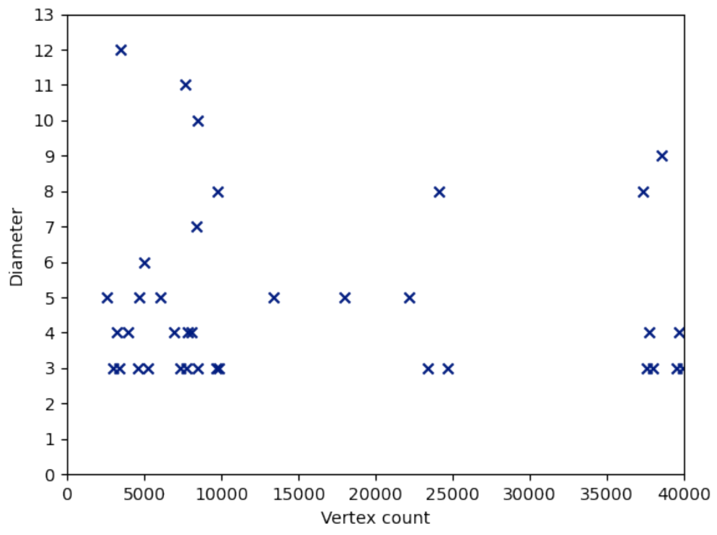

Figure 4.3.: The diameters plotted against the number of segments. Each marker represents one instance.

Density

Graphs with relatively few edges, called *sparse graphs*, and many edges, called *dense graphs*. In this section, we introduce a measurement of how dense a graph is and analyze the *density* of the competition instances.

A graph with $|V|$ vertices has at most $|V| \cdot (|V| - 1)/2$ edges. For a graph $G = (V, E)$, we define its density $d(G)$ as the number of edges relative to the maximum possible.

$$\begin{aligned} d(G) &= |E| / \frac{|V| \cdot (|V| - 1)}{2} \\ &= \frac{2|E|}{|V| \cdot (|V| - 1)} \end{aligned}$$

The density $d(G)$ of a graph is always in $[0, 1]$. We call a graph G sparse if its density is at most 0.1, following [LM01].

The Table 4.3 gives an overview over the density of graphs grouped by instance type.

Instance type	$d(G)$ average	$d(G)$ variance
reecn	0.227	$6.572 \cdot 10^{-4}$
sqrp	0.491	$2.902 \cdot 10^{-3}$
sqrpecn	0.464	$1.606 \cdot 10^{-3}$
visp	0.100	$2.970 \cdot 10^{-3}$
vispecn	0.105	$2.035 \cdot 10^{-3}$

Table 4.3.: The average and variance of density for each instance type.

The only instances that fulfill the condition of a density 0.1 are visp instances. These instances have dense subgraphs, however, which are only losely connected. Therefore, these graph do not fulfill the random graph assumption of [LM01]. We do not investigate this approach further.

Skewness

A graph is called *planar* if it has a plane drawing. For planar graphs, efficent coloring algorithms which require few colors are known. The *skewness* of a graph is a metric that indicates how far a graph is from being planar, with planar graphs having a skewness of 0. In this section, we investigate the skewness of the competition instances.

The idea is to partition the intersection graph into plane subgraphs, which can trivially be colored with six colors, and it is known that they can be colored with four colors [G$^+$08]. If this is possible without requiring to build too many subgraphs, a coloring with few colors can be reconstructed from the colorings of these subgraphs.

The *skewness* $s(G)$ of a graph G is the minimum number of edges that need to removed for the graph to be planar. Let G be a planar graph and f the number of faces in a plane drawing of G. Assuming that G is connected, the Euler formula states the following relationship between the graph and the number of faces:

$$|V| + f = |E| + 2$$

Each face is bounded by at least three edges, and each edge is adjacent to at most two faces. This gives the following relation between edges and faces:

$$3f \leq 2m$$

It follows that for each planar graph, the following inequality holds:

$$|E| \leq 3|V| - 6$$

If this inequality is not true, at least $|E| - 3|V| + 6$ edges need to be removed to make it true. This is a lower bound for the skewness of a graph:

$$s(G) \geq |E| - 3|V| + 6$$

The smallest minimum skewness across all instances belongs to the instance rvispecn2615 with a minimum skewness of 153644, and the average minimum skewness is 259789672.893. Due to the high minimum skewness, we do not investigate this approach further.

Vertex cover

Let $k \in \mathbb{N}$ a natural number we call the *parameter*, n the size of the problem instance. A *fixed parameter tractable* (FPT) problem is one that, for some function f and constant $c \in \mathbb{R}$, can be solved in $\mathcal{O}(f(k) \cdot n^c)$ time [DF99]. This means that problem is efficently solvable for small k.

Let k by the size of a vertex cover of a graph $G = (V, E)$. [FGK11] shows a proof that the graph can be colored in $\mathcal{O}((k^{k+1} + |E|) \cdot |V|)$ time.

This algorithm requires a vertex cover, however, and calculating one is also an NP-complete problem. A vertex cover can be calculated with another FPT algorithm which is also parameterized by the size of the vertex cover [GN07]. Because we already require the vertex cover size to be small for the coloring algorithm, this adds no new requirement. We use the FPT algorithm instead of heuristics in this section because we are no longer interested in actual vertex cover sizes, we only want to show that this approach is infeasible.

The algorithm answers whether there is vertex cover of size at most k for a graph G, and if so, calculates one. If there is a vertex v with degree $d(v) > k$, then it has to be part of the vertex cover. Assuming it was not, all of its neighbors would have to be in the vertex cover to cover all edges between v and its neighbors. The vertex v has more than k neighbors, which is more than is still allowed to be taken into the cover. Therefore, we add v to the cover under construction and apply the same step to the graph $G' = G - v$ with the limit of $k' = k - 1$ vertices for the cover of the remaining graph. If $k' = 0$ but G' still contains edges, we know that there is no valid vertex cover for this graph with size k.

Figure 4.4 shows the lower bounds on the vertex covers for all instances.

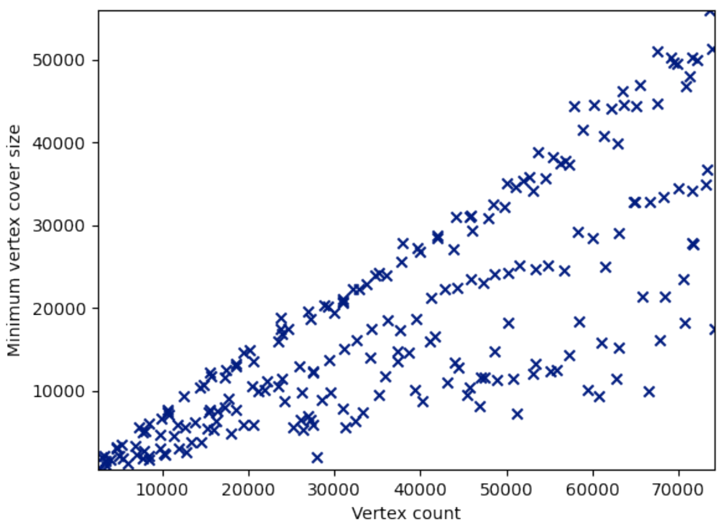

Figure 4.4.: The lower bound of the vertex covers plotted against the number of segments. Each marker represents one instance.

Using this coloring approach remains feasible up to $k \approx 10$ on our computers. The minimal lower bound of the vertex cover size over all competition instances is 499. Therefore, we do not investigate this approach further.

4.2. Instance type statistics

In this section, we analyze each instance type in more detail. First, we take a look at the -ecn modifier, which takes a geometric graph as its input and returns a modified version with slightly extended segments. Then, we show the analysis of the sqrp, visp and reecn instances, including their modified variations.

4.2.1. -ecn modifier

-ecn instances are created by modifying an existing instance. These instances are generated by extending the segments of an existing instance, following up on the motivation that the edge coloring problem for planar graphs [CN90] can be transformed into a plane partitioning problem by extending each edge.

Generation

The -ecn generation takes some instance and extends every segment slightly in both directions. This does not move the original endpoints of the segment, instead two new endpoints are created. After that, a few new random segments are added randomly.

This modifier uses the parameters $c > 1$, the scaling factor, and $p \in [0, 1]$, the chance for each segment between two points to be added after extending all existing segments. Let p_1p_2 be a segment. Then the new, modified segment is $p'_1p'_2$ with

$$p'_1 = p_2 + \frac{c+1}{2} \cdot (p_1 - p_2)$$
$$p'_2 = p_1 + \frac{c+1}{2} \cdot (p_2 - p_1)$$

This moves both points away from each other. If p_1 and p_2 are at distance d to each other, the points p_1' and p_2' are at distance $d' = d \cdot c$ to each other. This modification splits all endpoints with a degree greater than 0 in the representation into multiple, where each new point is only has degree 1.

Then, for each pair of points (p_1, p_2) which has no connecting segment, a new segment is added with probabilty p. This reduces the number of endpoints with degree 1.

Vertex cover of endpoints

The -ecn-modifier heavily increases the number of vertices while leaving the number of segments unchanged, we expect the size of the vertex cover to increase. In this section, we compare the sizes of vertex covers of -ecn and non-ecn-instances.

There are two trivial upper limits on the size of the vertex cover, the number of vertices and the number of edges in the graph. A vertex cover is a subset of vertices, and can therefore not be larger than the set of vertices. Because each edge requires only one of its endpoints to be in the vertex cover, one possible way to construct a vertex cover is to iterate over each edge uv, choose one of its endpoints u or v, and add it to the cover. This constructs a vertex cover that has at most one vertex per edge.

Given that we expect the size of the vertex cover with the size of the representations, which is why we do not provide absolute numbers for the vertex covers, but relative ones comparing to the number of segments and endpoints.

Table 4.4 It shows that the vertex covers of -ecn-instances are smaller relative to the number of points in the instance than non-ecn-instances, but larger relative to the number of segments.

Instances	rel. to #points	rel. to #segments
-ecn instances	50.645%	72.977%
other instances	82.896%	1.407%

Table 4.4.: A comparison of relative vertex cover sizes between -ecn and other instances.

Let $G = (V, E)$ be a graph where every vertex has degree 1. The sum of degrees over all endpoints is twice the number of edges, and because all degree are 1, this graph has twice as many vertices as edges $|V| = 2|E|$. An optimal vertex cover of this graph contains exactly one endpoint of each edge, which results in a vertex cover of size $|E| = |V|/2$. This explains why the size of the vertex cover relative to the number of points for -ecn instances is roughly 50%. The ednpoints of all extended segments have degree 1 until the additional segments are added. These additional segments

This variation also breaks the original motivation for the search of these vertex covers, namely that endpoints are a good source of large independent sets. Because most endpoints of -ecn-instances have a low amount of incident segments, this is not the case anymore.

4.2.2. sqrp instances

sqrp instances are generated by placing points randomly in a square and connect them randomly with a bias towards longer connections.

An example for a sqrp instance is the rsqrp4637 instance shown in Figure 4.5 and an example for a sqrpecn instance is the rsqrpecn8051 instance shown in Figure 4.6.

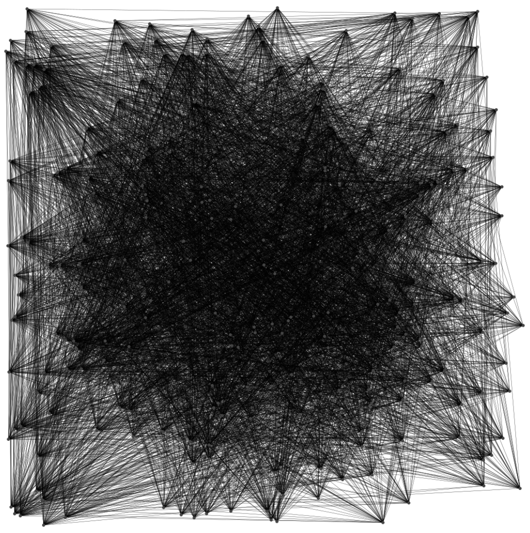
Figure 4.5.: The competition instance rsqrp4637.

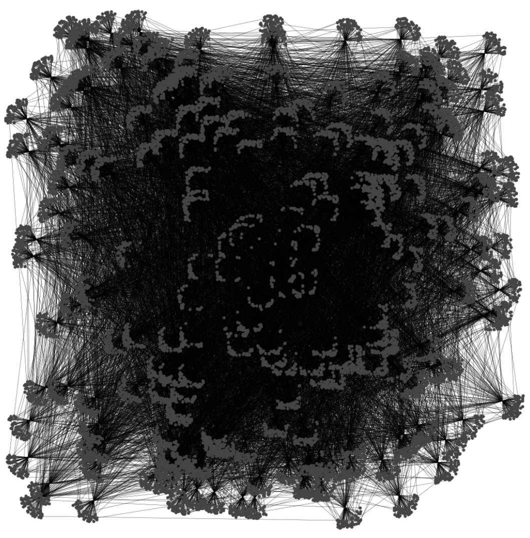
Figure 4.6.: The competition instance rsqrpecn8051.

Generation

All sqrp instances were generated with the following scheme with the input variables $k \in \mathbb{N}, c \in (0,1], p \in (0,1]$. First, generate a set of k random points inside a square. The side length and position of the square in the 2D-plane does not matter. Then, sort all of the $k(k-1)/2$ possible segments and only keep the longest $\lfloor c \cdot (k(k-1)/2) \rfloor$. From the remaining segments, add each one independently with a chance of p.

The expected number of segments in a sqrp instance is $p \cdot \lfloor c \cdot (k(k-1)/2) \rfloor$.

Instance parameters

From a competition instance, we can infer the values of k, c, and p. The parameter k is the number of points given in the representation. For the parameter c, we find the shortest segment s in the representation and find how many possible segments are longer than s.

We calculate the parameter p with the number of segments $|S|$ as $p = |S|/(c \cdot (k(k-1)/2))$. This works well under the assumption that the number of segments $|S|$ is close to its expected value.

Table 4.5 shows the parameters for a few sqrp instances calculated this way. The values for c and p are rounded to the nearest multiple of 0.05.

Instance name	k	c	p
rsqrp7320	471	0.30	0.80
sqrp10642	471	0.50	0.20
sqrp41955	533	0.50	0.60
sqrp43759	742	0.50	0.30
sqrp73525	1031	0.25	0.55

Table 4.5.: The value of the generation parameters k, c, p for some sqrp instances.

Intersection angles

During the analysis of the slopes, we concluded that a uniform random variable $X \sim U(0, \pi/2)$ is a good model intersection angles in general. In this section, we compare the actual distribution of intersection angles in sqrp instances to the theoretical one X.

Figure 4.7 shows a comparison between the uniform distribution X and the actual cumulative distribution of three sqrp instanecs.

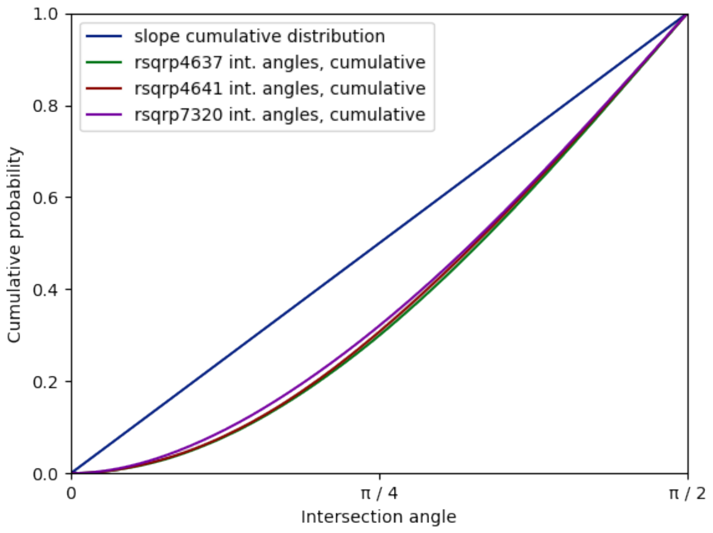

Figure 4.7.: The cumulative distribution of the intersection angles of the instance rsqrp4637, rsqrp4641, and rsqrp7320, and the uniform distribution $U(0, \pi/2)$.

This shows that for sqrp instances, intersection are more likely to be closer to $\pi/2$. This observation makes intuitive sense, as segments with similar slopes need to be close together to intersect, while segments with greatly differing slopes do not.

4.2.3. visp instances

The visp instances are random straight-line visibility graphs in polygons. This section provides the analysis of the visp and vispecn instances.

An example for a visp instance is the rvisp3499 instance shown in Figure 4.8 and an example for a vispecn instance is the vispecn7028 instance shown in Figure 4.9.

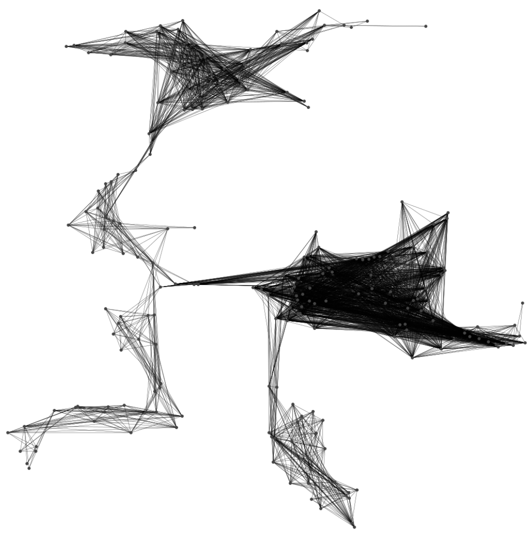

Figure 4.8.: The competition instance rvisp3499.

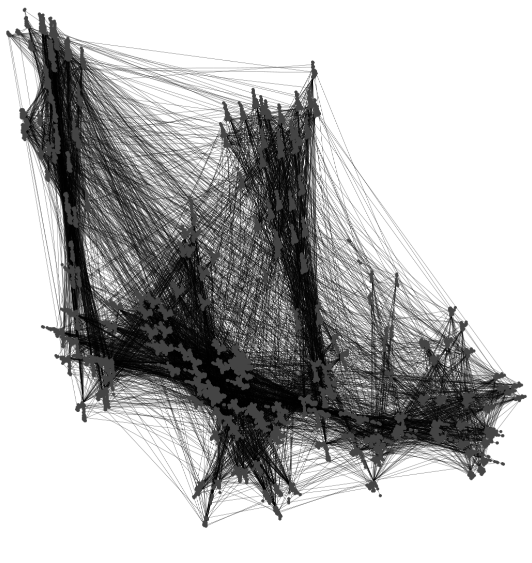

Figure 4.9.: The competition instance vispecn7028.

Generation

All visp instances were generated with the following scheme. Choose some polygon in the plane. This polygon may not self-intersect, but it may contain holes. Generate a set of random points inside the polygon. For all pairs of two points, if the segment between them does not intersect the boundary the polygon, add it.

Exact polygon reconstruction

It is impossible to reconstruct the original polygon with certainty, because there are multiple polygons that can result in the same representation. A simple example of this is in Figure 4.10, where the same representations arise from different polygons.

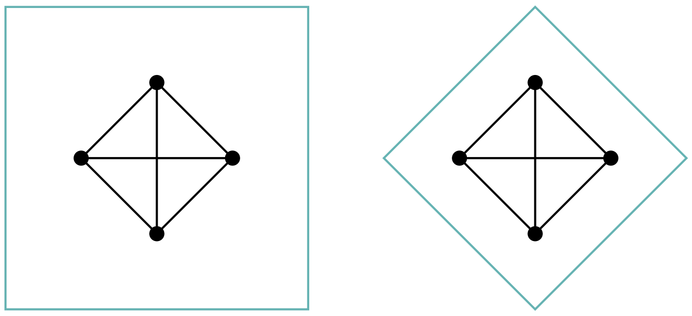

Figure 4.10.: The same representations can arise from different polygons.

For the following reconstruction methods, we assume that the intersection graph to the given representation has only one component. This may not be true for the given instances, but if it has multiple components, each of them may be examined separately because each component may be colored separately.

Because it is impossible to reconstruct the original polygon, we construct a polygon that matches the visp instance as precisely as possible. Formally, we construct a possible polygon for the generation that has minimal area.

To get this polygon, we first construct a plane geometric graph that has the same boundary. For each pair of intersecting segments p_1p_2 and p_3p_4, add a new point p' to the geometric graph. Then, replace the intersecting segments with the four new segments p_1p', $p'p_2$, p_3p', and $p'p_4$.

For each point, sort its incident segments clockwise. Start the construction of the polygon by adding the point p_1 furthest to the left. Then, add the point p_2 which is adjacent to p_1 and furthest up. For $p_i, i \geq 2$, add the segment that is next in the sorted segment list of p_i after the segment $p_{i-1}p_i$. Once the point p_1 is reached again, the polygon construction is complete.

This polygon has the minimal area. Assuming that it is not, there should be at least one point that may be removed. Because the polygon may not contain holes, the point has to be removed from its border. Every single point on the border belongs to a segment or a point, which means that removing it would make this point or segment impossible to generate from this polygon.

This construction method requires the calculation of all intersection points. Even though the segment endpoints of the original instances are always at integer coordiantes, the intersection points may not be. Additionally, the polygon can be used to speed up the calculation of the intersection graph. Because this reconstruction requires this calculation beforehand, we can not use this advantage.

Approximate polygon reconstruction

Motivated by the problems of an exact polygon reconstruction, we present an approximate polygon reconstruction that does not require any intersection calculation before the reconstruction.

The plan is to subdivide the bounding box into a grid of rectangles and determine for each rectangle whether it is intersecting with the instance or not. To eliminate the intersection checks between these rectangles and the segments of the instance, place k sample points uniformly distributed along each segment. A rectangle intersects the instance if one of these sample points lies within this rectangle.

For the formal construction, we consider the axis-aligned bounding box of the instance $R = (P, S)$ with minimal area. Let $(x_{\min}, y_{\min})$ be the lower left corner of the bounding box, and w the width and h the height of the bounding box. Then we choose a polygon resolution $r = 2^k, k \in \mathbb{N}$ and segment approximation precision a.

First, we translate the instance by $(-x_{\min}, -y_{\min})$ so that its lower left corner $(0, 0)$. Then, we scale it such that both its width and height are r. Let P' be the set of points used to approximate the polygon. For each segment p_1p_2, we add a set of evenly spaced points on the segment, $\{p_1 + (p_2 - p_1) \cdot (t - 1)/(a - 1) \mid t \in \mathbb{Z}_a\}$, to P'. For each square with sidelength 1 with its lower left corner on the integer grid, check if there is at least one point inside of it. If so, add it to the approximate polygon.

Figure 4.11 shows an approximate polygon reconstruction of instance visp26405 with $a = 32$ sample points per segment and a resolution of $r = 64$. Figure 4.12 shows the original instance for reference.

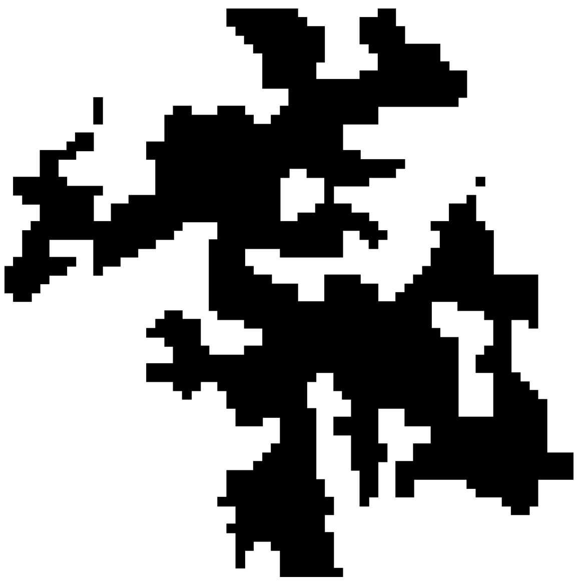

Figure 4.11.: An approximate polygon reconstruction of the competition instance visp26405.

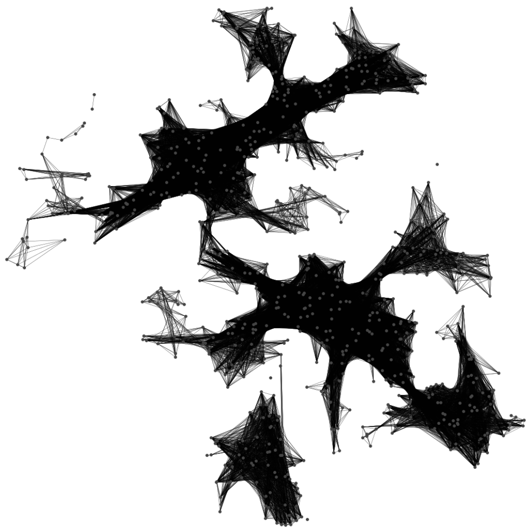

Figure 4.12.: The competition instance visp26405.

4.2.4. reecn instances

All reecn-instances have the -ecn modifier, and there are no competition instances without this modifier. These instances are general random graphs embedded into the plane and then modified by the -ecn modifier.

An example for a reecn instance is the reecn3382 instance shown in Figure 4.13.

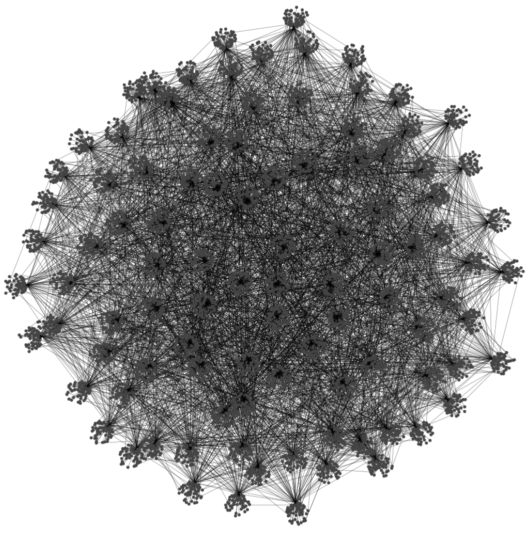

Figure 4.13.: The competition instance reecn3382.

Generation

To generate a reecn instance, generate a random graph $\mathcal{G}(n, p)$ with n vertices and embed it into the plane using the Kamada-Kawai-Layout [KK$^+$89]. Then, apply the -ecn-modifier to it.

For each pair of vertices $(v_1, v_2) \in V^2$, the Kamada-Kawai-Layout attempts match the distance of the points p_1, p_2 representing these vertices as closely as possible to the distance of v_1, v_2 in the graph $G = (V, E)$ to embed. An example of a Kamada-Kawai-Layout for a random graph with 50 vertices and edge probabilty 0.3 is shown in Figure 4.14.

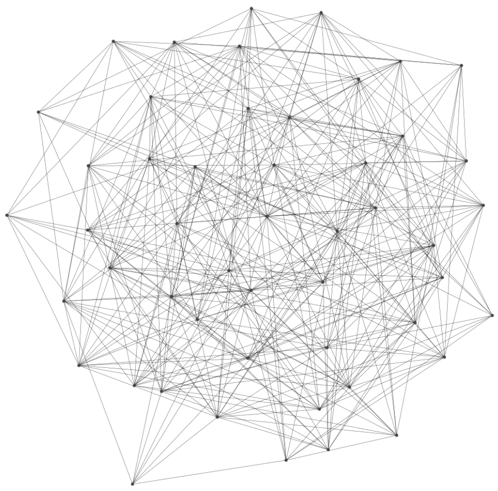

Figure 4.14.: A random graph with 50 vertices embedded with the Kamada-Kawai-Layout. The implementation for generating this embedding is from the networkx library for Python3.

5. Partitioning geometric graphs into plane subgraphs

This chapter presents and analyses the different coloring methods. There are two major components to coloring these instances, representation heuristics and general coloring algorithms, which we discuss in this chapter.

5.1. Representation heuristics

In this section, we introduce the representation heuristics used to color the competition instances.

5.1.1. General ideas

Here, we note a list of short ideas arising from the observations made in Chapter 4.

Independent sets from endpoints

For a graph $G = (V, E)$, we call a set of vertices $V' \subseteq V$ *independent* if there are no edges between any two vertices in V'. In a proper coloring of G, the number of vertices with same color is independent for each color used.

We already noted in Chapter 4 that segments with a common endpoint rarely intersect. Therefore, we know that the vertices of these segments in the intersection graph are an independent set in most cases.

Color distribution

In chapter 4, we made the observations that the slopes are approximately uniformly distributed, and that pairs of segments with similar slopes intersect less frequently than pairs of segments with a greater difference in slopes.

From these two observations, we come to the assumption that each color appears roughly equally often in an optimal coloring of the intersection graph. Each color then colors a set of segments with similar slopes.

Intersection angles

We already noted that pairs of segments with similar slopes are not as likely to intersect as segments with greatly differing slopes. Therefore, we use similar slopes as a heuristic to find large sets of slopes that do not intersect each other.

5.1.2. Partitioning

Instead of solving the coloring problem for the entire graph, we want to subdivide the graph into smaller ones that are solvable more efficiently and then combine these solutions to a solution on the entire graph. This section covers partitioning strategies and how well the coloring algorithms perform when combined with these partitioning strategies.

Partitioning with edge cuts

One way to partition a graph G is to partition its vertices V into two sets, color both induced subgraphs, and then combine these colorings to coloring of the complete graph.

For a partition V into V_1, V_2, we call the edges E_{cut} between the cut. The two best cases for a partition V_1, V_2 of vertices V are:

- There are no edges between the vertices of V_1 and V_2.

  $$E_{\text{cut}} = \emptyset$$

  In this case, both sets can be colored independent of each other. The same set of colors may be used for both sets of vertices, resulting in the final coloring having $\min(|c(V_1)|, |c(V_2)|)$ colors.

  If there are very few edges in the cut E_{cut}, the coloring can be combined using a local search method resolving any conflicts arising from these edges. Alternatively, the graph from V_1 is colored first, and V_2 considers the coloring of the vertices of V_1 when coloring its vertices.

- There is an edge between every vertex of V_1 and V_2.

  $$|E_{\text{cut}}| = |V_1 \times V_2|$$

  This reduces the number of edges to consider by $|V_1| \cdot |V_2|$. No color from one set may be used in the other one, resulting in the final coloring with $|c(V_1)| + |c(V_2)|$ colors.

  This strategy still works when most, but not all edges between the vertices of V_1 and V_2 are present. The fewer edges there are, the less effective this approach becomes.

Partition by slope

This section covers partitioning vertices such that ones with similar slopes end up in the same partition, as well as resulting properties of such a partition.

For this partition strategy, we first sort all vertices by their slope. Then, to form k partitions, let the i-th partition be the vertices from index $|V| \cdot ((i-1)/k) + 1$ to $|V| \cdot (i/k)$ in this order.

This approach uses edge cuts, which works well when there are a lot or very few edges in the cut. This partitioning approach has these optimal cases with few edges between adjacent partitions with index i and $i+1$, and many edges between partitions with index i and $i + k/2$. However, this partitioning results in the worst case inbetween. Because of this, we do not investigate this partitioning approach further.

Partitioning visp instances

The visp instances offer an intuitive way to partition them with vertex cuts. Figure 5.1 shows such a partitioning. The red lines mark the manual cuts, where each segment intersecting them is part of the cut. The blue lines are the segments in the cuts, and the green lines are the remaining segments.

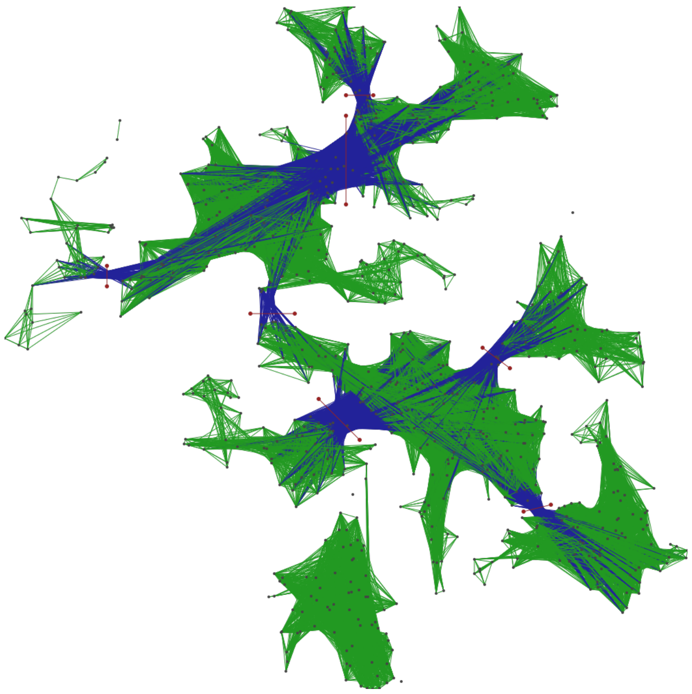

Figure 5.1.: A manual partition of the competition instance visp26405.

This manual partition is used to test coloring algorithms before creating a partitioning algorithm. We first colored all blue segments simultaneously because the vertices of different cuts are connected in some cases. Then, we color each component of green segments.

This partitioning approach required more colors with our coloring methods. Because of this, we do not investigate this partitioning approach further.

5.2. Coloring algorithms

In this section, we introduce the coloring algorithms we use to color the competition instances. We provide motivations and theoretical time complexities for these algorithms.

5.2.1. Sequential coloring

Sequential coloring algorithms color each vertex once in some order. This general idea makes them generally fast, because they only treat each vertex once. Because of this, these algorithms are already widely studied, for example in [Bré79, MMI72, KM04, Cul92].

All sequential algorithms have the same general structure. As long as at least one vertex has no assigned color, the algorithm chooses one uncolored vertex and assigns the vertex some color that no vertex in its neighborhood has. Selecting a good heuristic to select the vertex to color in each iteration is the main challenge when adapting the generic sequential Algorithm 2.

Algorithm 5.1: Generic sequential coloring

```
   Input: Graph $G = (V, E)$
   Output: Coloring $c : V \to \mathbb{N}$
1  init($G$);
2  forall $v \in V$ do
3      $c(v) = \perp$;    /* $\perp$ is a placeholder for vertices without a color */
4  while $\exists v \in V : c(v) = \perp$ do
5      $v \leftarrow$ selectVertex($G, c$);
6      $c(v) \leftarrow$ selectColor($v, G, c$);
7  return $c$
```

These algorithms run in $\mathcal{O}(T_{\texttt{init}}(G) + |V| \cdot (T_{\texttt{selectVertex}}(G) + T_{\texttt{selectColor}}(G)))$, where $T_{\texttt{algo}}(G)$ is the worst case time complexity of the respective algorithm `algo` when applied to the graph G.

Note that the vertex selection algorithm `selectVertex` may only choose a vertex that is not colored, but it may make its decision based on the graph and the current, incomplete coloring.

Greedy coloring

This section describes a simple, well-known sequential method to color a graph $G = (V, E)$, which we simply call greedy coloring.

This algorithm chooses any order $\sigma : \mathbb{Z} \to V$ during the `init` subroutine. There are no requirements for this order, and if the data structure that stores the graph implicitly provides such an order, may only take $\mathcal{O}(1)$ time.

During the ith call to `selectVertex`, the vertex $\sigma(i)$ is returned. This vertex is then assigned the minimum color that no vertex in its neighborhood has.

DSATUR

The DSATUR algorithm is defined in [Bré79]. This algorithm is a good heuristic for sequential coloring algorithms and is still used to motivate or as part of modern algorithms, for example in [Cul92, FGT16, San12].

DSATUR tries to identify the vertex which has the most limited options for the color that can be assigned to it, and color it first. To quantify how limited the options of a vertex are, the paper [Bré79] defines the saturation $s(v)$ of a vertex v as the number of unique colors in its neighborhood:

$$s(v) = |c(N(v)) \setminus \{\perp\}| \tag{5.1}$$

For this variation of the greedy algorithm, the `selectVertex` algorithm choose the vertex with the highest saturation. If multiple vertices are tied for the highest saturation, return any with the highest degree among them. An addressable priority queue, like pairing heaps[SV87], can accomplish this efficiently. The initialization of this data structure happens in `init`.

`selectColor` may not choose a color that is in the neighborhood of the selected vertex v, but it may make its decision based on the graph and the current, partial coloring. The most common choice for this, which is also presented in [Bré79], is to choose the smallest color that is not in the neighborhood of v.

$$\texttt{selectColor}(v, G, c) = \min(\mathbb{N} \setminus \{c(w) \mid vw \in E, c(w) \neq \perp\}) \tag{5.2}$$

If the graph is stored in a way that allows to iterate over all incident edges of v in $\mathcal{O}(d(v))$, for example as an adjacency list, then each edge is inspected exactly twice when calling this algorithm once for each vertex. This gives a time complexity of $\mathcal{O}(d(v))$ for a total time complexity of $\mathcal{O}(|V| + |E|)$ for all calls to this function during one coloring.

DSATUR variations

From the assumption that each color appears approximately equally often, two variations for `selectVertex` arise. The first one first attempts to give the vertex v a color of an already colored vertex w which corresponds to the segment with the most similar slope to the segment corresponding to v. We define the similarity of two slopes as their intersection angle

$$\min(|s_v - s_w|, \pi - |s_v - s_w|)$$

where smaller values mean that more similar slopes. For $s_v = s_w$, this value is 0. Every already colored vertex is tried in a non-decreasing order of this value. If none of their colors is not in the neighborhood of v, it is assigned a new color. This adds a factor of $|V| \log(|V|)$ to the runtime of `selectColor` for this algorithm.

The second one attempts to balance the number of occurrence of each color, which means that for the coloring c, the value of e should be minimal, where e is the difference between the number of occurrence of the most and least used color.

$$\begin{aligned} o(x) &= |\{v \mid c(v) = x\}| \\ e &= o(\max_{x \in C}(o(c))) - o(\min_{x \in C}(o(c))) \end{aligned}$$

[?] outlines a good heuristic to achieve this. `selectColor` always chooses the color that is already used, but is not in the neighborhood of vertex to color v and is used the least so far. If no such color exists, it is assigned a new color.

Slope order, randomized

Another sequential algorithm is the slope order approach. This one colors the vertices in the Increasing order of their respective slopes in the representation R of the graph G. Sorting the vertices requires $\mathcal{O}(|V| \log(|V|))$ time for `init`, after which each call to `selectVertex` can be handled in $\mathcal{O}(1)$. Together with the version of `selectColor` which always chooses the minimal possible color, the total time for this algorithm is $\mathcal{O}(|V| \log(|V|) + |E|)$.

Let $\sigma : \mathbb{Z}_{|V|} \to V$ be the slope order in which the vertices are colored. A circular shift of this order σ by $n \in \mathbb{N}$ is a new order σ_n, defined as

$$\sigma_n : i \mapsto \begin{cases} \sigma(i+n) & i + n \leq |V| \\ \sigma(i + n - |V|) & \text{otherwise.} \end{cases}$$

For the slope order approach, this is equivalent to the order of the instance when rotated by the slope of the vertex $\sigma(n+1)$ in the direction of decreasing slope. When comparing the two circular shifts σ_n and σ_{n+1}, we observe that the difference in colors used is usually zero or one. For any circular shift, the number of colors used does not deviate a lot from its average over all instances. Therefore, calculating a coloring for every circular shift, which adds a factor of $|V|$ to the time complexity, is not worth the effort. Instead, we choose k random values for the shift n and calculate a coloring for each of them. We abbreviate this coloring algorithm to SOR.

This approach relies on a segment representation for the intersection graph. If only the intersection graph is given, constructing a representation is a known $\exists\mathbb{R}$-complete problem [Mat14].

5.2.2. Coloring by endpoints

The coloring algorithm presented in this section does not require the intersection graph $G = (V, E)$ of the representation $R = (P, S)$. This removes the $\mathcal{O}(|S|^2)$ overhead for the naive intersection calculation or the $\mathcal{O}((|S| + |E|) \cdot \log(|S|))$ overhead for the Bentley-Ottmann algorithm [BO79]. Because this coloring algorithm does not use the intersection graph, we talk about coloring the segment of the geometric graph instead.

First, we assume that for each point $p \in P$ in the representation $R = (P, S)$, all incident segments of p do not intersect each other. We assign each points p a different color. Then, we assign each segment p_1p_2 the color of one of its endpoints p_1 or p_2.

The only remaining problem is how to choose the endpoint for each segment such that lowest number of colors possible are used. This is equivalent to finding the smallest vertex cover on the representation. We have already examined this problem in Chapter 4.

Last, we need to handle intersecting segments with the same endpoints p. For each point $p \in P$, we sort all incident segment clockwise in $\mathcal{O}(d(p) \log(d(p)))$ time. If two of the to p incident segments intersect, they are next to each other in this sorted list. Therefore, we can now find intersecting segments in $\mathcal{O}(n)$. When we find an intersection between two segments s_1 and s_2, we assign one of them a new color. We already observed that segments with common endpoints intersect rarely, which means that this method does not add many additional colors.

5.2.3. Random coloring

A random coloring uses each color approximately equally often. Because we already assume that the optimal color does so as well, we hope that a random coloring is close to a good local minimum that we can then find with a local search method.

Random coloring with conflicts

The first strategy creates a coloring with k colors. k is fixed. The coloring algorithm assigns each vertex one of the k colors with equal probability for each color.

This is most likely going to result in an improper coloring. These colorings can still be used for local search algorithms, because some of them do not require a proper coloring, only one that is close to a local minimum.

Random coloring without conflicts

The second strategy creates a random coloring without conflicts. When coloring a vertex v, it assigns a random color that is already assigned to some vertex, but not assigned to any neighbor of v. If no such color exists

The motivation for this second variant of random colorings are local search algorithms that do require a proper coloring as a starting point.

5.2.4. SAT-solver

Because most problems can be formulated as SAT instances, methods for solving SAT problems are extensively studied. We formulate our problem as a SAT problem and use the kissat solver [BFFH20] due to its outstanding performance in the 2020 SAT competition [FHI+21].

SAT solvers solve the decision problem whether an assignment $a : U \to \mathbb{B}$ exists that satisfies all clauses C, and if so outputs one such assignment. A *SAT formulation* of the coloring problem for a graph G with at most k colors is a SAT instance (U, C) that only has a satisfying variable assignment if the graph G is colorable with at most k colors. In such a case, the coloring $c : V \to \mathbb{Z}_k$ is reconstructable from the variable assignment t.

SAT formulation

The paper [Vel07] provides an overview of well-known SAT formulations of the coloring problem. We use the *muldirect* formulation.

This formulation uses one variable for each pair of a vertex v and a color c.

$$U = \{x_{(v,c)} \mid v \in V, c \in \{1, \ldots, k\}\}$$

In order to guarantee that a complete coloring is constructed, each vertex needs to have at least one possible color.

$$C_{\text{complete}} = \{x_{(v,1)} \vee \cdots \vee x_{(v,k)} \mid v \in V\}$$

In order to guarantee that a proper coloring is constructed, the vertices on each edge may not have the same color. To achieve this, we add a clause

$$C_{\text{proper}} = \{\bar{x}_{(u,c)} \vee \bar{x}_{(v,c)} \mid uv \in E, c \in \mathbb{Z}_k\}$$

The set of clauses used is the union of the two sets that formulate the requirements $C = C_{\text{complete}} \cup C_{\text{proper}}$. For each vertex v, we choose any color c such that the variable $x_{v,c}$ representing that this color may be chosen is true:

$$c(v) \in \{c \mid a(x_{(v,c)}) = \text{true}\}$$

Search strategy

Binary search is a well-known algorithm for finding the first value fulfilling some condition in a linear, monotone search space. We use it to find the lowest number of colors possible such that the graph is colorable.

When running the kissat SAT solver, it only terminates when it either finds a satisfying variable assignment or a set of unsatisfiable clauses. Especially when close to the chromatic number of the graph, it just runs out of memory after a long calculation. To prevent this, we limit the calculation time for each color to three minutes. After that time limit, we interrupt the calculation and assume that the current instance is unsatisfiable.

With this limitation, the SAT solver almost always terminates either because it found a satisfying variable assignment or because the limit was hit. The time limit termination takes a considerable bit longer than most terminations by finding a satisfying variable assignment.

Motivated by the problem of binary search, we instead search from n down.

5.3. Coloring improvement algorithms

The algorithms in this section require both a graph $G = (V, E)$ and a valid coloring $c : V \to \mathbb{Z}_n$ as their input and return a coloring $c' : V \to \mathbb{Z}_{n'}$, which uses no more colors than the input coloring c.

5.3.1. Iterative greedy coloring

This section presents the iterative greedy algorithm from [Cul92]. This includes the original idea of repeatedly applying a sequential coloring algorithms and using the coloring of the last iteration as a heuristic for the current one, as well as adaptations to this exact use case where representation heuristics may be used.

Proof of improvement

Consider a greedy coloring algorithm that colors the vertices in some order σ, and each vertex is colored by giving it the smallest color that does not belong to a colored neighbor. Then there is at least one order σ such that this algorithm uses the minimal amount of colors necessary to color this graph.

The proof we present here is a little stronger, showing that for any coloring, there is some order of coloring the vertices such that the resulting coloring requires an equal or lower amount of colors than the coloring from which the order was constructed. Let $c : V \to C$ be the initial coloring. Then, choose the order $\sigma : \{1, \ldots, |V|\} \to V$ such that if two vertices have the same color in this order, all vertices between them have this color as well.

$$\forall i, j, k \in \{1, \ldots, |V|\}, i < j < k : c(\sigma(i)) = c(\sigma(k)) \Rightarrow c(\sigma(i)) = c(\sigma(j))$$

This order has the vertices of each color in a continuous *block*. Now, apply the algorithm to this order. Each block consists of independent vertices, which implies that the greedy algorithm only ever finds lower colors in the neighborhood. Therefore, a vertex in the ith block can not have a color greater than i. This especially means that no vertex has a color greater that the number of blocks, which means that this coloring does not use more colors than the one used to construct the order σ. Using an optimal coloring for the construction of σ proves that such an order exists.

Algorithm description

The iterative coloring algorithm takes any order to start. It then greedily colors the graph, creates a new order from this coloring with above construction algorithm, and then repeats this process. This leaves two degrees of freedom when choosing the order from the coloring.

The first is the order of the vertices within some block of colors. Experimental results from [Cul92] show that ordering vertices from non-increasingly by their degree is a good idea, and they reasoned that this is to be expected because DSATUR has a similar idea. We examine two variations of the algorithm. In the first one, we use the original ordering of non-Increasing degrees. The second one is ordering the vertices non-decreasingly by the slope of the segment representing them, motivated by the slope order algorithm.

The second is the order of color blocks themselves. Multiple approaches are presented in [Cul92], but not all of them are used by us to generate colorings. The ones that are used are:

- **Increasing size:** This strategy orders the blocks non-decreasingly by the number of vertices with this color.
- **Decreasing size:** This strategy orders the blocks non-increasingly by the number of vertices with this color.
- **Reverse order:** This strategy orders the blocks in decreasing order of the color they represent.
- **Random order:** This strategy randomizes the order of the blocks.

These strategies are run in parallel and the one with the best result is chosen. When at least one strategy reduces the number of colors used, the greatest decrease in the number of colors is the best result. When one iteration of this algorithm does not reduce the number of colors used with all strategies, another metric has to be used to measure progress. The paper [Cul92] presents a metric that encourages large independent sets to form by taking the sum of all assigned colors. More precisely, for each color $c \in \mathbb{Z}_n$, the number of vertices

$|V_c|$ with this color c is counted and the value $|V_c| \cdot c$ is added to the value of the metric. Smaller values are preferable.

This algorithm terminates when no progress was made in one iteration. As this may happen after a very large number of iterations, which in turn takes considerable time, a limit on the number of iterations is imposed as well.

5.3.2. Tabu search

This section describes how the tabu search, a variation of the local search, performed on these instances. An overview of tabu search approaches is in the survey [GH06]. These algorithms are mainly variations on the algorithm `tabucol`, first described in [HdW87].

In [GH06], a search strategy is defined as a triple of the search space that is explored, the neighborhood function defining to what solutions we may take a step from the current one, and the evaluation function that has to be minimized.

The search space is the set of all possible solutions we consider. We start at one of these solutions. Then we move to through the search space by repeatedly moving from our current solution to one in its neighborhood. The evaluation function approximates how far our current solution is from an optimal one.

Tabucol

The search space of the original `tabucol` [HdW87] is the set of all complete colorings for the graph, including improper ones, with at most k colors. A step consists out of recoloring one vertex v from its original color c to c'. The objective function to minimize is the number *conflicts*, formally the number of edges vw with the same color on each end $c(v) = c(w)$. When the number of conflicts is 0, we have found a valid solution.

The first start point is a random coloring with conflicts, motivated by the fact that the random coloring is most likely close to local minimum with few colors. This has a high number of conflicts, requiring more progress to found a valid solution. This version is one used in the paper on `tabucol` [HdW87].

The second start point is taking some proper coloring with $k + 1$ colors and recoloring all vertices with color $k + 1$ to another random color. This starts with a low number of conflicts, but it is likely difficult to escape a local minimum.

k-fixed partial legal

The search space of the k-fixed partial legal [Mor96] strategy is the set of all proper colorings for the graph that use at most k colors, including incomplete ones. A step consists out of giving an uncolored vertex v a color c, and set all neighbors of v with the color c to uncolored. The objective function to minimize is the sum of degrees of uncolored vertices:

$$\sum_{\substack{v \in V \\ c(v) = \bot}} d(v)$$

When the sum of degrees of uncolored vertices is 0, we have found a valid solution, assuming that all vertices with degree 0 are colored.

The first start point is a random coloring without conflicts using $k + 1$ colors, motivated by the fact that the random coloring is most likely close to local minimum with few colors. Then, we set all vertices with color $k + 1$ to uncolored. This leaves us with an incomplete, but proper coloring with k colors.

The second start point is taking some proper coloring with $k + 1$ colors and setting all vertices with color $k + 1$ to uncolored. This starts with a low sum of degrees of uncolored vertices, because rarer colors do not have many vertices to take sum of degrees of, and more frequent colors usually color vertices with low degrees. As with `tabucol`, it is likely difficult to escape a local minimum.

5.4. Evaluation

In this section, we evaluate the coloring algorithms we introduced in this chapter. The evaluation only uses a representative subset of the competition instances. For a full list of colorings, see Table A.2 through A.6 in the appendix.

Coloring by endpoints

The main motivation for the coloring by endpoints algorithm is that it does not require the intersection graph. Before the competition started, a set of test instances with up to 1000000 segments were released. The creation of this algorithm is mainly motivated by instances of this size where calculating the intersection graph would be difficult. Because the competition instances only have up to 75000 segments, coloring by endpoints loses its use case.

Table 5.1 compares the sizes of vertex covers on the representation, which are a good lower bound on the number of colors required by the coloring by endpoints algorithm, compared to the number of colors required by the greedy algorithm on the intersection graph.

Instance name	rep. vertex cover	greedy
reecn7847	5551	165
reecn73116	51917	727
sqrp7730	231	197
sqrp73525	649	787
sqrpecn33659	25444	751
rvisp7648	277	88
visp73369	521	455
vispecn33280	23652	418

Table 5.1.: Representation vertex covers compared to greedy colorings of this intersection graph for select instances.

While this method manages to beat the greedy algorithm in some cases, sequential coloring algorithms with better heuristics, like DSATUR, are better on all instances. The main point of this coloring algorithm is that it can efficiently produce proper and complete colorings, but producing such colorings is not a problem on the competition instances.

DSATUR and variations

Next, we evaluate the performance of the DSATUR algorithm and its variations. Table 5.2 shows two things. First, like on general graphs [Bré79], the DSATUR algorithm beats the greedy algorithm by a wide margin. Second, the original DSATUR algorithm and the DSATUR slope variation usually return similar results, while the DSATUR balanced variation does consistently require more colors.

Instance name	DSATUR	DSATUR balanced	DSATUR slope	greedy
reecn7847	99	115	98	165
reecn73116	435	554	437	727
sqrp7730	120	137	120	197
sqrp73525	432	540	448	787
sqrpecn33659	492	572	480	751
rvisp7648	52	61	53	88
visp73369	311	364	–	455
vispecn33280	278	330	–	418

Table 5.2.: The colors required by DSATUR and its variations in comparison.

In our further evaluation, we only consider the original DSATUR algorithm. The balanced variation requires more colors, and the slope variation adds a massive $|V|\log(|V|)$ time factor.

Slope order, randomized

In this chapter, we introduced the SOR algorithm. This sequential algorithm manages to color the intersection graphs with fewer colors in most instances, as Table 5.3 shows.

Instance name	DSATUR	SOR
reecn7847	99	88
reecn73116	435	392
sqrp7730	120	102
sqrp73525	432	337
sqrpecn33659	492	424
rvisp7648	52	60
visp73369	311	260
vispecn33280	278	302

Table 5.3.: Representation vertex covers compared to greedy colorings of this intersection graph for select instances.

Local search

Both local search algorithms examined, the original tabucol and the k-fixed partial legal strategy, fail to reduce the number of colors required. This is true when starting with a random coloring which uses as many colors as a DSATUR or SOR coloring, as well as when starting with a valid DSATUR or SOR coloring.

The experimental results in [HdW87, Mor96] investigating these two algorithms used smaller instances for their tests. Both methods use a metric that has to be zero for a new coloring to be found. In the survey [GH06], the importance of the *aspiration criterion* is stressed. The aspiration criterion allows a tabu step to be taken if it reduces the metric of the tabu search algorithm to zero.

These three facts combined explain why the tabu searches work so poorly on the contest instances. These methods do not make continuous progress, but have a binary outcome, either finding a solution with fewer colors or not. This algorithm does not scale well to large instances because the search space gets larger and 'flatter', making it more difficult to step to a solution from a local minimum. This is supported by the importance of the aspiration criterion, which assists in making this step. The fact that it is so important shows that the tabu search already has trouble making this step on smaller graphs.

Iterative greedy

We use the iterative greedy algorithm to improve coloring acquired with the DSATUR and SOR algorithm.

The experimental results from [Cul92], show that this algorithm typically drastically reduces the number of colors needed in the first few iterations, followed by a lot of iterations with no progress. Our use of this algorithm exhibits the same behavior.

We evaluate two variants of this algorithm. The first one improves DSATUR colorings and uses non-increasing degree order for its blocks, the other one improves SOR colorings and uses slope order for its blocks. We evaluate both variants motivated by the fact that the DSATUR algorithm performs better on some instances, while the SOR algorithm performs better on others. The comparison between these two iterative greedy variations, as well as the results of the initial solutions, are in Table 5.4.

Instance name	DSATUR	DSATUR iter	SOR	SOR iter
reecn7847	99	99	88	87
reecn73116	435	424	392	376
sqrp7730	120	120	102	100
sqrp73525	432	426	337	335
sqrpecn33659	492	471	424	418
rvisp7648	52	52	60	57
visp73369	311	–	260	255
vispecn33280	278	–	302	272

Table 5.4.: A comparison between the iterative greedy strategy with DSATUR and SOR as start points.

kissat SAT solver

Here, we discuss the results acquired by the kissat SAT solver [BFFH20]. As Table 5.5 with the example of some reecn instances, kissat was able to tie the result of the iterative greedy approach on small competition instances, but it does not manage to achieve the same result on bigger instances.

Instance name	DSATUR iter	SOR iter	kissat
reecn3382	95	88	88
reecn21194	207	207	216
reecn56737	273	273	280
reecn63079	318	317	328

Table 5.5.: Results of kissat SAT solver for coloring these graphs compared to the best results acquired.

6. Conclusion

In this thesis, we introduced and examined the problem of partitioning geometric graphs into plane subgraphs. We showed that this problem and some subproblems limiting the geometric graph are NP-complete, and evaluated the feasibility and performance of different approaches using the CH:SHOP 2022 competition instances as benchmarks.

We motivated and introduced a new coloring method for intersection graphs of geometric graphs, SOR, and compared it to well-known coloring methods, like DSATUR [Bré79]. Additionally, we adapted this new coloring algorithm to improve the existing iterative greedy method [Cul92]. The most successful coloring algorithms in our evaluation are the DSATUR algorithm followed by the iterative greedy algorithm, and the SOR algorithm followed by the adapted version of the greedy algorithm.

The results acquired with our methods require in sum over all instances $\sim$13% more colors than the results of top team of the competition ”Shadoks”. With this result, we placed 11th out of 40 teams. The top four teams of the competition are invited by the organizers of the CH:SHOP 2022 competition to present their result as part of the CG Week 2022, which is set to happen on June 07. - 10. 2022.

Open problems

From the results of this thesis, multiple new ideas for further analysis of intersection graphs arising from geometric graphs.

Random graph models

Aside from the random graphs $\mathcal{G}_{n,p}$ first introduced in [ER$^+$60, ER61], other random graph models have been proposed to model problems. An example are power-law random graphs [ACL01], where the distribution of degrees follows a power law. Each generation for one of the five instance types is a random graph model that can be studied further.

This can also be extended to other random generation models for geometric graphs. A paper which already provides an outlook what a very general analysis of random geometric graphs looks like is [YY84], which provides a formula for the probability of two segments intersecting if their bounding boxes intersect.

SAT formulation

The paper [Vel07] provides a survey of SAT formulations on general graphs. This thesis has shown that representation heuristics can be better than well-studied coloring algorithms on general graphs, like SOR compared to DSATUR [Bré79]. This motivates the search for a SAT formulation of the intersection graph coloring problem that uses the representation.

One idea is to build on the idea of [AHK$^+$17], which considers the maximum number of plane spanning trees there are in a complete geometric graph. To find these trees, the paper [AHK$^+$17] proposes to find a set pairwise intersecting segments. For each of these segments, one of its endpoints then connects to all points to the left and the other to all points to right of this segment. Removing the condition that these trees need to be spanning trees and formulating the choice of tree generating segments as a SAT or ILP problem might result in a better coloring method.

Local search

Similar to the motivation for SAT formulations using the geometry of the representation, we motivate local searches making use of the representations. One idea is to create a metric for progress that favors moving towards solutions where segments with similar slopes have the same color, following up on the observation that an optimal solution most likely has this feature.

FPT coloring algorithms

The graph class Seg provides a new possible parameter for fixed-parameter tractable algorithms, namely the number of slopes used in the representation. [KN90] shows that for a representation with at most k different slopes, the clique problem on the intersection graph $G = (V, E)$ can be solved in $\mathcal{O}(|V|^k)$ time. This raises the question if there is an FPT algorithm for coloring intersection graphs of geometric graphs.

Bibliography

[ACL01] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental mathematics*, 10(1):53–66, 2001.

[ACL11] Eric Angel, Romain Campigotto, and Christian Laforest. Analysis and comparison of three algorithms for the vertex cover problem on large graphs with low memory capacities. *Algorithmic Operations Research*, 6(1):56–67, 2011.

[ACL12] Eric Angel, Romain Campigotto, and Christian Laforest. Implementation and comparison of heuristics for the vertex cover problem on huge graphs. In Ralf Klasing, editor, *Experimental Algorithms*, pages 39–50, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[AHK$^+$17] Oswin Aichholzer, Thomas Hackl, Matias Korman, Marc van Kreveld, Maarten Löffler, Alexander Pilz, Bettina Speckmann, and Emo Welzl. Packing plane spanning trees and paths in complete geometric graphs. 124:35–41, August 2017.

[BFFH20] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[BO79] Bentley and Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, 1979.

[Bré79] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, apr 1979.

[BW80] Bentley and Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, C-29(7):571–577, 1980.

[Cav19] Dario Cavallaro. Hamiltonicity and the computational complexity of graph problems. Bachelor Thesis, July 2019.

[CCL13] Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discrete & Computational Geometry*, 50(3):771–783, September 2013.

[CG09] Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: Extended abstract. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC ’09, page 631–638, New York, NY, USA, 2009. Association for Computing Machinery.

[CN90] Marek Chrobak and Takao Nishizeki. Improved edge-coloring algorithms for planar graphs. *Journal of Algorithms*, 11(1):102–116, 1990.

[Cul92] Joseph Culberson. Iterated greedy graph coloring and the difficulty landscape. 1992.

[Dai80] David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. *Discrete Mathematics*, 30(3):289–293, 1980.

[DF99] R. G. Downey and M. R. Fellows. *The Basic Definitions*, pages 23–28. Springer New York, New York, NY, 1999.

[DL08] François Delbot and Christian Laforest. A better list heuristic for vertex cover. *Information Processing Letters*, 107(3):125–127, 2008.

[ER$^+$60] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[ER61] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1):261–267, 1961.

[FGK11] Jiří Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, May 2011.

[FGT16] Fabio Furini, Virginie Gabrel, and Ian-Christopher Ternier. An improved DSATUR-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, 69(1):124–141, November 2016.

[FHI$^+$21] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Sat competition 2020. *Artificial Intelligence*, 301:103–572, 2021.

[G$^+$08] Georges Gonthier et al. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[GH06] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006. Part Special Issue: Anniversary Focused Issue of Computers & Operations Research on Tabu Search.

[GJMP80] M. R. Garey, D. S. Johnson, Gary L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Alg. Disc. Meth.*, 1(2):216–227, June 1980.

[GJS74] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, page 47–63, New York, NY, USA, 1974. Association for Computing Machinery.

[GN07] Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, mar 2007.

[Gol85] Martin Charles Golumbic. Interval graphs and related topics. *Discrete Mathematics*, 55(2):113–121, 1985.

[HdW87] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, December 1987.

[Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[KK$^+$89] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[KM94] J. Kratochvil and J. Matousek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62(2):289–315, 1994.

[KM04] Adrian Kosowski and Krzysztof Manuszewski. Classical coloring of graphs. *Contemporary Mathematics*, 352:1–20, 2004.

[KN90] Jan Kratochvíl and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 031(1):85–93, 1990.

[Kra91] Jan Kratochvíl. String graphs. ii. recognizing string graphs is np-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991.

[LM01] Manuel Laguna and Rafael Martí. A grasp for coloring sparse graphs. *Computational Optimization and Applications*, 19(2):165–178, 2001.

[Mat14] Jirí Matousek. Intersection graphs of segments and $\exists\mathbb{R}$. *CoRR*, abs/1406.2636, 2014.

[MMI72] David W. Matula, George Marble, and Joel D. Isaacson. Graph coloring algorithms. In RONALD C. READ, editor, *Graph Theory and Computing*, pages 109–122. Academic Press, 1972.

[Mor96] Craig Morgenstern. Distributed coloration neighborhood search. *Discrete Mathematics and Theoretical Computer Science*, 26:335–358, 1996.

[Qua21] Valentin Quapil. Upward and upward-planar drawings with limited slopes. Bachelor Thesis, September 2021.

[San12] Pablo San Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39(7):1724–1733, 2012.

[SV87] John T. Stasko and Jeffrey Scott Vitter. Pairing heaps: Experiments and analysis. *Commun. ACM*, 30(3):234–249, mar 1987.

[Vel07] Miroslav N. Velev. Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 135–142, 2007.

[YY84] Mark C.K Yang and Chung-Chun Yang. Assessment of the intersection probabilities of random line segments and squares. *Computer Vision, Graphics, and Image Processing*, 26(3):319–330, 1984.

7. Appendix

A. Coloring tables

This section contains tables with ten coloring strategies. To fit these table on the pages, the names of coloring strategies are abbreviated, see Table A.1.

Algorithm name	Abbreviation
DSATUR balanced	db
DSATUR slope	ds
DSATUR	dsat
greedy	gr
Iterative greedy, SOR	i_sorb
Iterative greedy, SOR start	i_sor
Iterative greedy, DSATUR	iter
kissat	k_sor
slope order	so
slope order, randomized start	sor

Table A.1.: Algorithm name abbreviations.

Table A.2 contains the reecn colorings, Table A.3 contains the sqrp colorings, Table A.4 contains the sqrpecn colorings, Table A.5 contains the visp colorings, and Table A.6 contains the vispecn colorings. The numbers marked in bold in these tables are the best coloring we achieved for the instance. If a cell is marked with a ”–”, we do not have coloring for that instance with that strategy.

Instance name	db	ds	dsat	gr	i_sorb	i_sor	iter	k_sor	so	sor
reecn3382	103	92	94	146	**88**	89	95	**88**	94	93
reecn3988	94	82	77	132	**73**	74	81	74	77	76
reecn6910	163	148	150	242	**137**	**137**	145	143	147	145
reecn7847	115	98	99	165	**87**	**87**	99	88	88	88
reecn9674	188	162	164	266	**151**	152	163	157	160	159
reecn11799	219	205	200	321	**187**	**187**	197	193	196	194
reecn12588	188	159	160	255	**143**	**143**	159	145	150	147
reecn15355	242	207	207	342	**191**	192	204	198	201	199
reecn16388	225	184	192	306	165	**164**	189	171	174	170
reecn17244	239	200	199	326	**176**	177	193	184	183	182
reecn18615	215	176	176	296	**155**	**155**	173	156	158	**155**
reecn21194	273	233	225	374	**207**	**207**	223	216	215	213
reecn21946	281	224	231	361	201	**200**	222	207	207	205
reecn23484	268	221	218	358	**191**	**191**	211	194	193	193
reecn23945	296	243	252	407	**219**	**219**	248	228	230	227
reecn25913	348	308	302	490	**275**	277	298	296	297	293
reecn27494	290	232	236	382	**203**	**203**	230	208	211	208
reecn29395	324	258	267	426	227	**226**	259	236	237	234
reecn31126	358	297	293	494	267	**266**	293	280	281	278
reecn32569	390	325	331	547	**300**	**300**	324	320	320	317
reecn34307	426	371	367	601	339	**335**	362	360	361	359
reecn36204	442	391	386	635	**354**	357	383	376	379	376
reecn37593	360	282	285	468	249	**247**	276	256	256	253
reecn39476	384	304	313	507	270	**268**	302	280	283	278
reecn41237	484	429	419	690	**385**	388	416	416	415	412
reecn42891	503	448	447	737	415	**412**	439	436	438	435
reecn44299	477	414	410	668	375	**374**	404	396	400	395
reecn45823	509	439	441	729	**403**	**403**	430	432	435	431
reecn47287	459	386	384	626	**340**	341	377	356	361	356
reecn48602	478	402	398	662	**361**	364	394	384	383	380
reecn50133	463	366	363	609	**316**	**316**	355	328	330	328
reecn51526	471	396	400	642	350	**349**	391	364	364	361
reecn53304	441	340	342	579	**291**	292	334	300	303	299
reecn54867	438	345	339	575	**295**	**295**	336	300	302	300
reecn56737	419	326	329	540	**273**	**273**	319	280	278	278
reecn58325	535	445	449	744	**402**	406	436	424	428	424
reecn59950	494	389	391	634	**333**	**333**	384	344	344	342
reecn61436	424	315	316	539	**265**	**265**	306	272	274	271
reecn63079	477	373	368	621	318	**317**	365	328	332	329
reecn64798	603	503	501	838	**457**	**457**	494	–	488	484
reecn66642	554	450	451	735	**398**	**398**	437	–	416	414
reecn68262	560	455	448	743	394	**393**	438	–	413	410
reecn69952	583	471	472	784	**419**	420	460	–	440	438
reecn71622	542	433	429	706	**367**	**367**	414	–	379	377
reecn73116	554	437	435	727	376	**375**	424	–	392	392

Table A.2.: The colorings of the reecn instances.

Instance name	db	ds	dsat	gr	i_sorb	i_sor	iter	k_sor	so	sor
sqrp7730	137	120	120	197	**100**	**100**	120	–	102	102
sqrp9831	170	143	144	250	**118**	**118**	141	–	121	**118**
sqrp10642	183	152	158	266	**128**	**128**	156	–	130	129
sqrp12451	185	161	161	276	**130**	**130**	156	–	133	131
sqrp15532	205	184	183	317	**145**	**145**	180	–	150	**145**
sqrp18603	231	196	200	339	**157**	**157**	195	–	160	158
sqrp20166	236	204	200	338	**158**	**158**	198	–	159	**158**
sqrp20602	238	193	198	339	161	**160**	193	–	163	161
sqrp23758	257	227	225	379	**175**	**175**	223	–	177	**175**
sqrp26930	291	246	243	426	**194**	**194**	241	–	195	195
sqrp28863	294	252	255	434	**199**	**199**	249	–	202	200
sqrp30973	314	261	258	449	**200**	201	255	–	201	201
sqrp32894	302	250	247	404	**192**	**192**	242	–	196	**192**
sqrp34718	336	280	283	486	**219**	**219**	279	–	225	**219**
sqrp36053	320	268	271	435	**203**	**203**	264	–	206	**203**
sqrp37978	329	277	278	446	**209**	**209**	270	–	210	**209**
sqrp39917	331	280	276	451	**213**	**213**	272	–	214	**213**
sqrp41955	333	284	272	453	210	**208**	269	–	211	210
sqrp43759	357	293	293	499	**228**	229	287	–	234	231
sqrp45933	354	285	293	471	**222**	223	286	–	228	223
sqrp47890	345	283	291	441	214	**212**	285	–	216	213
sqrp49981	444	358	356	633	275	**274**	344	–	277	275
sqrp51004	390	319	318	521	243	**242**	312	–	249	244
sqrp53087	406	334	325	560	256	**255**	321	–	261	258
sqrp53628	438	362	347	628	**272**	273	345	–	276	275
sqrp55426	406	332	325	550	256	**254**	320	–	257	255
sqrp56833	413	351	331	570	**259**	260	328	–	265	263
sqrp57865	450	381	369	630	**278**	279	362	–	282	**278**
sqrp60119	480	403	393	732	**299**	**299**	382	–	301	**299**
sqrp62212	423	359	352	556	**259**	260	348	–	263	**259**
sqrp63419	536	430	427	757	**324**	**324**	422	–	334	326
sqrp63650	445	364	357	578	**267**	**267**	352	–	275	268
sqrp65541	425	354	348	540	**258**	**258**	344	–	267	259
sqrp67451	487	395	398	668	**293**	**293**	392	–	299	294
sqrp69121	464	387	375	620	**281**	282	370	–	283	283
sqrp69435	471	384	377	626	**281**	282	375	–	287	284
sqrp70811	471	380	376	609	278	**277**	371	–	286	281
sqrp72075	466	381	380	609	279	**278**	374	–	285	280
sqrp73525	540	448	432	787	**335**	**335**	426	–	341	337
rsqrp4637	101	90	89	151	**78**	**78**	89	79	82	**78**
rsqrp4641	103	90	91	139	**74**	**74**	89	**74**	79	**74**
rsqrp7320	134	119	116	185	**94**	**94**	112	96	**94**	**94**
rsqrp14364	204	170	173	294	139	**138**	171	–	142	139
rsqrp23406	246	207	208	338	**164**	**164**	201	–	166	**164**
rsqrp24641	275	229	222	391	**181**	**181**	220	–	185	183

Table A.3.: The colorings of the sqrp instances.

Instance name	db	ds	dsat	gr	i_sorb	i_sor	iter	k_sor	so	sor
sqrpecn3020	152	131	138	202	126	**125**	135	–	132	130
sqrpecn3218	156	136	140	206	**130**	**130**	137	–	136	133
sqrpecn5276	190	168	173	262	162	**161**	172	–	168	165
sqrpecn8508	289	263	264	415	**242**	243	–	–	252	247
sqrpecn10560	329	295	291	446	**270**	**270**	290	–	278	275
sqrpecn10755	274	236	234	360	**215**	**215**	232	–	225	221
sqrpecn14854	407	364	364	545	332	**331**	356	–	342	339
sqrpecn15605	436	402	397	613	**367**	**367**	389	–	383	376
sqrpecn17186	393	340	340	519	**316**	317	331	–	332	325
sqrpecn17395	448	400	406	610	370	**366**	394	–	382	377
sqrpecn18520	445	397	401	603	372	**370**	388	–	384	383
sqrpecn19349	512	469	471	705	**430**	431	462	–	451	437
sqrpecn23715	575	511	504	762	**461**	464	497	–	475	469
sqrpecn23873	509	454	460	701	**416**	**416**	449	–	431	425
sqrpecn27255	518	443	453	705	**390**	391	440	–	409	397
sqrpecn29223	546	481	478	753	**435**	**435**	472	–	453	447
sqrpecn30017	501	445	443	683	**389**	**389**	428	–	406	402
sqrpecn30957	546	477	477	757	**420**	**420**	461	–	443	432
sqrpecn31026	557	493	484	767	**443**	**443**	477	–	482	460
sqrpecn32073	608	546	537	840	**497**	498	529	–	529	510
sqrpecn33659	572	480	492	751	418	**417**	471	–	428	424
sqrpecn35230	625	537	531	848	**474**	**474**	528	–	491	487
sqrpecn37744	644	580	586	915	517	**516**	563	–	536	531
sqrpecn39689	657	580	579	905	501	**499**	563	–	514	509
sqrpecn41897	665	589	576	923	499	**498**	559	–	520	509
sqrpecn44118	699	638	635	1005	549	**548**	612	–	562	559
sqrpecn45700	677	588	570	952	**495**	**495**	560	–	514	504
sqrpecn45811	692	577	569	923	**491**	**491**	558	–	513	502
sqrpecn48383	690	583	588	937	505	**503**	572	–	515	510
sqrpecn49763	683	606	613	1000	**518**	**518**	591	–	547	529
sqrpecn51856	735	626	619	982	537	**535**	600	–	558	545
sqrpecn52587	739	636	623	1032	547	**545**	612	–	567	560
sqrpecn54576	737	648	647	1036	**555**	557	625	–	595	574
sqrpecn56236	782	701	692	1129	608	**607**	681	–	629	625
sqrpecn57317	750	644	645	1023	**567**	**567**	624	–	608	584
sqrpecn58790	835	717	721	1144	**623**	624	704	–	638	636
sqrpecn61354	774	657	652	1034	**572**	**572**	628	–	597	590
sqrpecn62891	765	675	663	1068	**577**	579	646	–	610	596
sqrpecn65041	845	722	712	1204	**605**	**605**	698	–	619	616
sqrpecn67473	826	719	707	1145	590	**589**	680	–	613	605
sqrpecn69904	938	823	804	1351	**700**	703	791	–	718	711
sqrpecn71261	857	729	740	1160	**618**	621	–	–	652	633
sqrpecn71571	923	785	760	1241	672	**671**	–	–	692	682
sqrpecn73925	948	836	825	1327	729	**726**	–	–	770	753
rsqrpecn8051	240	220	216	342	198	**196**	213	208	211	206

Table A.4.: The colorings of the sqrpecn instances.

Instance name	db	ds	dsat	gr	i_sorb	i_sor	iter	k_sor	so	sor
visp26405	123	112	107	154	99	**98**	–	–	103	101
visp27975	83	68	64	89	**59**	60	–	–	63	61
visp29489	156	134	134	227	**131**	132	–	–	146	137
visp31334	123	104	105	152	**90**	**90**	–	–	91	**90**
visp32354	142	135	127	174	**98**	100	–	–	106	101
visp34162	181	169	167	271	**150**	152	–	–	160	158
visp35881	185	156	160	241	140	**139**	–	–	158	141
visp37336	188	177	168	281	164	**163**	–	–	175	170
visp38574	196	180	177	287	**149**	**149**	–	–	160	**149**
visp40191	151	133	**126**	217	**126**	**126**	–	–	139	131
visp41039	211	–	177	270	**150**	**150**	–	–	154	151
visp43093	176	–	152	228	**126**	**126**	–	–	137	128
visp44362	185	–	165	260	**157**	158	–	–	164	158
visp45466	158	–	137	229	**130**	132	–	–	145	139
visp46936	147	–	**114**	214	118	119	–	–	144	131
visp48558	198	–	171	309	**170**	**170**	–	–	177	177
visp50129	244	–	208	329	174	**173**	–	–	191	178
visp51133	159	–	129	194	108	**104**	–	–	123	112
visp53088	180	–	147	254	139	**137**	–	–	147	142
visp55158	187	–	167	255	**134**	135	–	–	142	139
visp57201	193	–	171	283	**166**	167	–	–	174	169
visp59449	186	–	154	252	150	**149**	–	–	178	157
visp60660	184	–	137	230	**121**	122	–	–	124	122
visp62685	178	–	**150**	259	151	152	–	–	168	156
visp64932	288	–	250	406	**225**	**225**	–	–	229	228
visp66498	195	–	144	227	144	**142**	–	–	162	152
visp68333	261	–	201	336	**178**	**178**	–	–	201	180
visp70702	219	–	184	306	**179**	**179**	–	–	183	182
visp71536	299	–	246	392	209	**208**	–	–	241	216
visp73369	364	–	311	455	**255**	**255**	–	–	279	260
rvisp3499	44	39	41	58	**38**	**38**	41	**38**	39	**38**
rvisp5013	67	58	**57**	95	60	61	**57**	65	69	66
rvisp7648	61	53	**52**	88	57	56	**52**	59	69	60
rvisp8404	69	66	60	93	**55**	56	59	–	60	57
rvisp8432	63	52	54	79	**47**	48	54	–	56	48
rvisp9770	77	72	69	111	**67**	**67**	69	–	74	69
rvisp10374	72	67	73	108	65	64	**63**	–	70	69
rvisp11339	112	93	94	141	**84**	**84**	93	–	90	**84**
rvisp12844	75	69	69	98	56	**55**	68	–	61	57
rvisp14562	89	80	79	127	**75**	**75**	79	–	87	81
rvisp15254	110	99	96	157	98	96	**95**	–	102	99
rvisp15474	130	117	116	174	98	**97**	115	–	104	98
rvisp20601	115	111	107	169	**101**	103	107	–	116	106
rvisp22145	166	149	150	213	**124**	**124**	149	–	131	125
rvisp24116	161	147	139	220	**131**	**131**	137	–	137	136

Table A.5.: The colorings of the visp instances.

Instance name	db	ds	dsat	gr	i_sorb	i_sor	iter	k_sor	so	sor
vispecn2518	83	–	76	98	**71**	**71**	–	–	85	76
vispecn5478	154	–	139	205	141	**138**	–	–	165	152
vispecn7028	165	–	153	218	156	**151**	–	–	183	168
vispecn10178	164	–	144	213	**141**	**141**	–	–	171	161
vispecn11917	208	–	189	261	**180**	181	–	–	226	201
vispecn13806	333	–	306	427	**279**	280	–	–	326	307
vispecn15912	266	–	**242**	380	252	251	–	–	283	281
vispecn16227	318	–	**312**	468	316	**312**	–	–	366	354
vispecn17665	369	–	342	526	342	**341**	–	–	392	376
vispecn19370	312	–	279	370	**259**	262	–	–	333	296
vispecn20413	402	–	**377**	605	380	380	–	–	439	422
vispecn25263	269	–	220	321	**209**	210	–	–	250	228
vispecn26025	338	–	284	404	280	**271**	–	–	323	301
vispecn26166	403	–	391	562	378	**377**	–	–	436	427
vispecn26914	296	–	259	366	**236**	243	–	–	272	264
vispecn27222	304	–	254	381	250	**245**	–	–	320	280
vispecn27480	262	–	217	320	214	**208**	–	–	242	235
vispecn27572	426	–	**391**	634	413	408	–	–	485	469
vispecn28567	408	–	360	519	339	**335**	–	–	373	370
vispecn31031	397	–	313	440	**295**	**295**	–	–	350	327
vispecn33280	330	–	278	418	**272**	**272**	–	–	363	302
vispecn35198	414	–	367	491	**332**	334	–	–	438	373
vispecn37349	503	–	451	662	434	**433**	–	–	499	497
vispecn39381	432	–	327	509	**319**	321	–	–	371	358
vispecn41599	484	–	446	675	432	**431**	–	–	485	472
vispecn43993	502	–	467	690	**443**	446	–	–	541	519
vispecn45672	407	–	335	476	321	**319**	–	–	399	372
vispecn46968	441	–	376	544	338	**335**	–	–	391	381
vispecn47378	423	–	**363**	544	373	368	–	–	425	415
vispecn48944	429	–	390	565	**363**	366	–	–	413	411
vispecn50715	424	–	359	522	344	**343**	–	–	444	384
vispecn53314	485	–	415	578	**352**	355	–	–	415	400
vispecn55775	454	–	379	563	**363**	**363**	–	–	414	406
vispecn58391	615	–	**509**	789	521	517	–	–	607	606
vispecn61049	478	–	**435**	703	449	452	–	–	525	521
vispecn63113	543	–	444	635	404	**399**	–	–	460	440
vispecn65831	698	–	595	916	592	**591**	–	–	671	647
vispecn67795	535	–	492	672	**452**	453	–	–	572	510
vispecn70501	693	–	622	935	**594**	599	–	–	704	683
vispecn71708	666	–	615	908	**582**	584	–	–	667	656
vispecn74166	547	–	477	708	453	**451**	–	–	512	498
rvispecn2615	50	**46**	**46**	67	47	47	**46**	–	52	51
rvispecn6048	98	90	91	126	**89**	92	90	–	108	96
rvispecn13421	267	231	241	302	207	**204**	215	–	257	226
rvispecn17968	271	233	246	347	239	235	**232**	–	286	260

Table A.6.: The colorings of the vispecn instances.