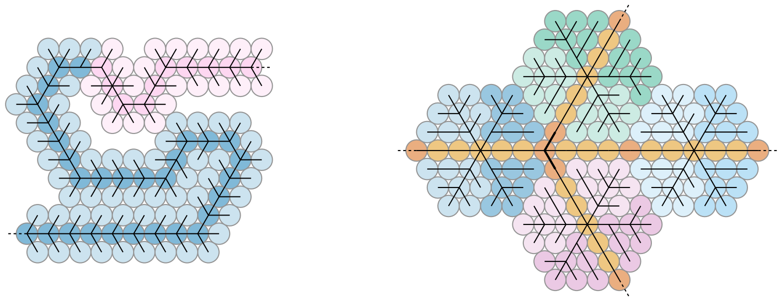
Weak Unit Disk Contact Representations and Colored Nearest Neighbor Graphs



Dissertation zur Erlangung des akademischen Grades
„Doktor der Naturwissenschaften“ (Dr. rer. nat.)

am

Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von

Jonas Cleve

Berlin, 2024

Betreuer & Erstgutachter:	Prof. Dr. Wolfgang Mulzer, Freie Universität Berlin
Zweitgutachter:	Prof. Dr. Martin Nöllenburg, Technische Universität Wien

Tag der Disputation:	8. November 2024

Abstract

Instances of geometric problems usually have both a concrete geometric and a more abstract representation. Given the geometric representation it is often easy to find the abstract one while the converse is not true: Given an abstract representation it can be challenging to find a geometric one or decide that no such representation exists. How challenging it is generally depends on the complexity of the given abstract representation. In this thesis we study two different problems with the aforementioned properties:

Weak Unit Disk Contact Representations. A unit disk contact representation (UDCR) of a graph is a set of interior-disjoint unit disks in the plane (each disk corresponds to one node) such that two disks touch if and only if their corresponding nodes have an edge. The notion of weak unit disk contact representations (weak UDCRs) weakens this condition and only enforces that two disks must touch if the corresponding nodes have an edge. If two nodes don't have an edge, their corresponding disks are still allowed to touch. The problem comes in two flavors: the first are graphs without embedding where the neighboring disks can be arranged in any order. Here we show that the problem is NP-hard for trees and provide a linear time algorithm for caterpillars, which are trees that become paths when all leaves are removed once. The second flavor are graphs with a fixed embedding where a given order of the neighboring disks in a weak UDCR must be respected. Here we show that the problem is already NP-hard for general caterpillars. We also show that we can decide in linear time whether a caterpillar has a weak UDCR that can be placed on a triangular grid and whose disks for the path of inner nodes are strictly x-monotone.

Colored Nearest Neighbor Graphs. In the second part of this thesis we look at a one-dimensional geometric problem. Here we are given a set of one-dimensional points and a list of line segments between neighboring points such that every point has at least one incident line segment. We then assign a non-empty set of colors to each point and for each assigned color create an edge in this color between the point and the closest point that also has this color. A valid assignment of colors to the points has two properties: First, every created edge corresponds to a line segment between the same endpoints and for every line segment there exists such an edge. Second, there can be no two edges with different colors between two points.

It is easy to see that such a valid color assignment always exists: select a unique color for each line segment and assign this color to both endpoints. In this thesis we answer the question of how many colors are needed for a given input and how long it takes to find a valid assignment. We show that for both one and two colors we can decide whether a valid assignment exists (and also find it) in linear time. To this end we make some structural observations that help us narrow down the possible color assignments that we need to look at. There are two defining structures responsible for increasing the necessary number of colors: local maxima which are line segments that are longer than both adjacent line segments, and small gaps which are missing line segments between two points such that the missing line segment is not longer than both adjacent line segments. We show that in the worst case the number of local maxima increases the number of colors logarithmically and the number of small gaps increases them linearly. Finally, we provide a dynamic program that, given an input and a number of colors, finds a valid color assignment with this number of colors or returns that no valid color assignment exists. The running time of this algorithm is exponential in the number of colors.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass alle verwendeten Hilfsmittel und Hilfen in der Arbeit angegeben sind und ich auf dieser Grundlage die Arbeit selbstständig verfasst habe. Die Arbeit wurde nicht in einem früheren Promotionsverfahren eingereicht.

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht.

Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Berlin, den 27. Juni 2024

Jonas Cleve

Acknowledgements

This thesis concludes several years of learning, teaching, and researching at Freie Universität Berlin. Many people have supported me during this time and helped me complete this work.

First, I would like to thank my advisor Wolfgang Mulzer for making it all possible in various ways. The classes he taught during my master's studies strengthened my interest in theoretical computer science and laid the groundwork for my PhD. I'm grateful for his guidance, supervision, and, most importantly, his trust in my abilities and perseverance. His valuable feedback has helped to improve this thesis. None of this would have been possible without his efforts to fund my position and provide opportunities for me to attend conferences, workshops, and research visits.

I want to thank Martin Nöllenburg for agreeing to be my second reviewer. He also brought my attention to the problem covered in the first part of this thesis. I enjoyed our work together on this problem and appreciate his invitation for a research visit in Vienna.

During my time in the theoretical computer science group, there were numerous people who made the PhD journey enjoyable—too many to thank individually. I will fondly remember our fun work trips, educational and interesting coffee rounds, and pleasant corridor chats. There are, however, some people I would like to thank separately:

I'm grateful to Kristin for our joint research on colored nearest neighbor graphs, without which the second part of this thesis wouldn't exist. Even more important was having her as a friend and sharing an office in the second part of my PhD. I also want to thank Nadja for sharing an office with me at the beginning of my PhD and always being willing to listen to my thoughts. My thanks go to Katharina for her feedback on the first part of this thesis and to Max for his insights into improving teaching. I will fondly remember Frank both as a teacher and as a colleague.

Finally, I thank my family for always supporting me and for the knowledge that they will continue to do so.

Contents

Frequently Used Mathematical Notation

Both Parts

$[1..i]$	The integer set $\{1, 2, \dots, i\}$.
2^A	The power set of the set A, i.e., $2^A = \{B \mid B \subseteq A\}$.
$\mathbb{N}$	All natural numbers $\{0, 1, 2, 3, \dots\}$.
$\mathbb{N}^+$	All positive natural numbers $\{1, 2, 3, \dots\}$.

Part I: Weak Unit Disk Contact Representations

$[A]^b$	All subsets of A of size b, i.e., $[A]^b = \{C \subseteq A \mid \lvert C\rvert = b\}$.
$1_i, \dots, 6_i$	Position $1, \dots, 6$ relative to a disk D_i, see Section 4.1.1.
$B(r, r')$	The branching gadget with $r, r' \geq 3$, see Definition 3.7.
$\mathscr{D}$	A disk set consisting of interior-disjoint unit disks; usually refers to a weak UDCR, see Definitions 2.1 to 2.3.
$\mathscr{D}_i$	The i-restricted weak UDCR of the weak UDCR $\mathscr{D}$, see Definition 3.2.
D_i	The unit disk in $\mathscr{D}$ corresponding to node v_i in the graph, see Definitions 2.1 to 2.3.
$\deg(v)$	The degree of node v, i.e., the number of adjacent nodes in the graph.
$\exists\mathbb{R}$	The complexity class containing all languages with a polynomial time reduction to the existential theory of the reals.
$F(B, p)$	The set of all free positions if all positions in $B \cup \{p\}$ are blocked, i.e., $F(B, p) = [1..6] \setminus (B \cup \{p\})$, see Definition 4.2.
$F_l(B, p),\ F_r(B, p)$	The contiguous set of free positions in $F(B, p)$ starting counter-clockwise and clockwise of p, respectively, see Definition 4.2.
G_i	The i-restricted caterpillar of the caterpillar G, see Definition 3.1.

Γ	The combinatorial embedding of a graph, i.e., the order of the adjacent nodes around each node, see page 10.
φ_i	The free angle of disk D_i in $\mathscr{D}_{i-1}$, see Definition 3.3.
$R(r)$	The rigid tree with radius r, see Definition 3.5. Also called chainable hexagon tree for $r \geq 3$, see page 32.

Part II: Colored Nearest Neighbor Graphs

$p \circ\!\!\longrightarrow q$	Point q is the unique nearest neighbor of p, see Definition 5.4.
$p \circ\!\!\xrightarrow{c} q$	Point q is the unique nearest neighbor of p among all points that have color c, and p itself also has color c, see page 85.
$p \circ\!\!\xrightarrow[\sigma]{c} q$	Point q is the unique nearest neighbor of p within the points that have color c assigned by σ (and p is among those points), see page 85.
B_i	A block, i.e., all consecutive points between two special edges, including one endpoint of each special edge, see Definition 6.6.
B_i^*	An exclusive block, i.e., all consecutive points between two special edges, excluding the endpoints of the special edges, see Definition 6.6.
$C_{\hat{c}}$	The set of $\hat{c}$ colors $\{1, \dots, \hat{c}\}$, see Definition 5.6.
$\mathbb{C}_{\hat{c}}$	The set of all subsets of $C_{\hat{c}}$ of sizes one and two, see page 91.
E_P	The neighbor edges of point set P, see Definition 5.10.
$\mathscr{G}(E), \mathscr{G}_E$	All gaps in E, i.e., all edges in E_P but not in E, see Definition 5.12.
$\mathscr{G}^-(E),\ \mathscr{G}_E^-$	All small gaps in E, i.e., all gaps that are not local maxima in E_P, see Definition 6.2.
$\mathscr{M}(E),\ \mathscr{M}_E$	All local maxima in E, see Definition 5.14.
$\mathscr{N}(P)$	The edges of $\mathscr{N}\mathscr{G}(P)$, see Definition 5.5.
$\mathscr{N}(P,\sigma)$	The edges of $\mathscr{N}\mathscr{G}(P,\sigma)$, see Definition 5.7.
$\mathscr{N}\mathscr{G}(P)$	The nearest neighbor graph (NNG) of point set P, see Definition 5.5.
$\mathscr{N}\mathscr{G}(P,\sigma)$	The colored nearest neighbor graph (CNNG) of point set P with color assignment σ, see Definition 5.7.

$\overline{R}(p,q)$	The restricted region, taking the interval from p to q, flipping it to the other side of p and then excluding p, see Definition 6.1.
$\overleftarrow{R}_E(e)$, $\overrightarrow{R}_E(e)$	The left / right restricted region of edge e with respect to input edge set E. For local maxima the restricted region is based on e, for small gaps it is based on the adjacent edges, see Definition 6.3.
$\mathscr{S}(E)$, $\mathscr{S}_E$	All special edges in E, i.e., all local maxima $\mathscr{M}_E$ and all small gaps $\mathscr{G}_E^-$, see Definition 6.5.
$\mathrm{swap}_{a,b}(c)$	A function that returns a if $c = b$, b if $c = a$, and c unchanged otherwise, see Definition 6.4.

Chapter 1
Introduction

When talking about geometric problems, it is generally the case that, in addition to the geometric representation, we can find a more abstract representation of the same situation. In this thesis we consider two problems which, very broadly, consider objects and whether the objects should be considered neighbors or not. The obvious abstract representation for these situations is a graph: it has a node for each object and an edge between two nodes if and only if the corresponding objects are considered to be neighbors. It is easy to obtain the graph, given a geometric representation, as the geometric representation determines how many objects there are and whether two of them are neighbors or not. However, the reverse is not generally true: Given a graph, we need to position the objects in some geometric space such that they are considered neighbors if and only if they have an edge. Finding a fitting geometric representation can be easily solvable or NP-hard, depending on various factors: These factors include the complexity of the problem itself and the complexity of the input graph. In addition, the problems are generally contained in the complexity class $\exists\mathbb{R}$: the position of each object can be described by a finite number of real values while the neighborhood relation can be described by a finite number of (in)equalities per object pair.

The problem of *weak unit disk contact representations* which has the aforementioned properties introduced in Section 1.1 and covered in detail in Part I. The second problem covered in this thesis does not completely fit the aforementioned properties. Here we already have some geometric information in the abstract representation. This problem of *colored nearest neighbor graphs* is introduced in Section 1.2 and covered in detail in Part II.

1.1 Weak Unit Disk Contact Representations

A *disk contact representation (DCR)* for a graph is a set of interior-disjoint disks in the plane (each disk corresponding to one node) such that two disks touch if and only if the corresponding nodes are adjacent. Koebe [Koe36] showed nearly a century ago that planar graphs are exactly the graphs that have a disk contact representation (with varying radii). Thus, it is possible to decide in linear time whether a graph has a DCR or not with a linear time planarity testing algorithm, e.g., one by Hopcroft and Tarjan [HT74], Boyer and Myrvold [BM04], or de Fraysseix, de Mendez, and Rosenstiehl [FMR06]. One drawback is that for general planar graphs the ratio between the largest and smallest radius in a DCR may be exponential in the number of nodes. Thus, Alam, Eppstein, Goodrich, Kobourov, and Pupyrev [Ala+14] consider *balanced* DCRs where the ratio between the largest and smallest radius is polynomial in the number of nodes. They show that certain graph classes, e.g., planar graphs with bounded maximum degree and logarithmic outerplanarity,[1] and trees, always admit balanced DCRs while others, e.g., planar graphs with bounded maximum degree and linear outerplanarity, sometimes don't.

Restricting the disks' radii further such that they are all equal introduces *unit disk contact representations (UDCRs)*. Breu and Kirkpatrick [BK98] showed that it is NP-hard to decide whether a given graph has a UDCR. The results were then improved by two groups at the same time. Klemz, Nöllenburg, and Prutkin [KNP15] (see [KNP22] for a full version) showed that the problem is already hard for outerplanar graphs. The linear time decision algorithm for caterpillars (trees that become paths after removing all leaves) claimed in [KNP15] was retracted in [KNP22] and remains a conjecture. They also showed that, if the embedding of the graph, i.e., the cyclic order of the nodes' neighbors, is fixed, the problem remains NP-hard for outerplanar graphs. This result was strengthened (independently) by Bowen, Durocher, Löffler, Rounds, Schulz, and Tóth [Bow+15] who showed that the problem with a given embedding is already NP-hard for trees.

Results. In this thesis, we relax the notion of UDCRs to *weak unit disk contact representations (weak UDCRs)* where we allow two disks to touch even if their corresponding nodes do not have an edge. Equivalently, we can say that a graph has a weak UDCR if a supergraph with the same node set and a superset of the edges exists that has a UDCR. In Section 3.1 we first provide an algorithm that finds a weak UDCR for caterpillars in linear time or decides that a given caterpillar does not have a weak UDCR. We first claimed this result in [Cle20], but the proof turned out to be incorrect. Here we show why and then provide a different proof for the same result. A more general algorithm for slightly weaker constraints has been presented by Bhore, Löffler, Nickel, and Nöllenburg [Bho+21a; Bho+21b]. The authors give a linear time algorithm to find weak UDCRs for lobsters (trees that become caterpillars after removing all leaves) such that the disk centers are restricted to the triangular grid and the positions of the disks of the backbone nodes are strictly x-monotone.[2] The backbone is the path that remains after removing all leaves of a caterpillar (or all leaves of a lobster, twice). Our

[1]A graph has outerplanarity k if it is planar and k is the smallest number such that iteratively removing all nodes incident to the outer face k times results in the empty graph.

[2]For some more details on their results see the first paragraph in Section 3.1.

Graph Class	Weak UDCRs	Weak Embedded UDCRs
Trees	NP-hard (Theorem 3.2)	↑ NP-hard
Caterpillars	$O(n)$ (Theorem 3.1)	NP-hard (Theorem 4.2)
Grid-Restricted, Strictly x-Monotone Caterpillars	↓ $O(n)$	$O(n)$ (Theorem 4.1)
Grid-Restricted, Strictly x-Monotone Lobsters	$O(n)$ [Bho+21b, Lemma 4]	—

Table 1.1: An overview of the results regarding weak unit disk contact representations. The hardness results also apply to all graph classes above and the algorithms also work for all graph classes below. This is indicated by the arrows. The result for grid-restricted, strictly x-monotone lobsters is set apart as it is not directly comparable with caterpillars and trees.

second result, presented in Section 3.2, is to show that it is NP-hard to decide whether a tree has a weak UDCR.

When looking at a weak UDCR of a graph, the order of disks around a parent disk is irrelevant. If, however, we want the disks to be ordered in a certain way, we call this a *weak embedded unit disk contact representation (weak embedded UDCR)*. In Section 4.1 we present a linear time dynamic programming algorithm that finds a weak embedded UDCR for a caterpillar such that all disk centers are restricted to the triangular grid and the positions of the disks on the backbone are strictly x-monotone. The backbone is the path that remains after removing all leaves of a caterpillar. Secondly, in Section 4.2 we show that it is already NP-hard to decide whether a caterpillar has a weak embedded UDCR.

In Table 1.1 we can see an overview of our results regarding different graph classes. For both NP-hardness results we also show that the problem is contained in $\exists\mathbb{R}$. This is the class of all problems which can be reduced in polynomial time to a Boolean formula of the form $\exists x_1, \dots, x_n \in \mathbb{R} : F(x_1, \dots, x_n)$. Here, F is a quantifier-free formula that allows logically combining equalities and inequalities of polynomials of the variables x_1 to x_n. We also show that the problem is in NP and thus becomes NP-complete if we only consider the grid-restricted problem.

1.2 Colored Nearest Neighbor Graphs

For the second problem discussed in this thesis, we start by talking about the *undirected nearest neighbor graph (NNG)* of a point set. This graph has the points as nodes and an undirected edge between two nodes if one of the two points is closer to the other than to any third point from the point set. Whether the point needs to be strictly closer or not differs in the literature. However, often some kind of general position assumption is made. This results in every point having exactly one other point strictly closer than all other points. See Paterson and Yao [PY92] or Mitchell and Mulzer [MM17, Section 32.1] for an overview of general results on NNGs or general proximity graphs. If we assign one or more colors to each of the points we can then compute the NNG for each color by only considering the points which have this color. We can then combine those graphs to obtain one *colored nearest neighbor graph (CNNG)*. This NNG variant was introduced by van Kapel [Kap14] in the context of puzzle generation. There, the task is not to compute the CNNG but the following: Given a set of points in the plane and a set of straight line segments between points, assign colors to the points such that the resulting CNNG is equal to the input. In their work [Kap14] and in the subsequent work by Löffler, Kaiser, van Kapel, Klappe, van Kreveld, and Staals [Löf+14] the authors are mainly interested in practical heuristics for simplifying a given drawing such that it can be approximated by a CNNG of few points.

A slightly different variant of the problem was then studied by Cleve, Grelier, Knorr, Löffler, Mulzer, and Perz [Cle+22]. The big difference is that each point is only allowed to have exactly one color, which means that the point set is partitioned into independent point sets whose NNGs are calculated independently. The question is then whether there exists such a partition into a given number of sets such that the combined NNGs represent the input segments. The first result is that for at least three colors it is NP-complete to decide whether such a partition exists even if the input has no crossings, i.e., no two line segments in the input cross in their interior. For two colors, it remains NP-complete in the general case but becomes solvable in polynomial time if the input has no crossings.

Results. In this thesis, we restrict ourselves to the one-dimensional case. We define CNNGs similarly to [Kap14] which means that we allow each point to have more than one color. The function that assigns the colors to the point is called a color assignment. If the same edge appears in the NNG of two or more different colors, the result is a *colored nearest neighbor multigraph (CNNM)* as the edge is counted for each color individually. Our goal, however, will be to find CNNGs and not CNNMs: we want to color the points such that each input edge appears in the NNG for exactly one color. Since we are restricted to one dimension our input consists of points from $\mathbb{R}$, and we also only allow edges between two consecutive points. In Chapter 6 we look at the problem for one and two colors. We obtain an algorithm that lets us decide in linear time whether a given point set can be colored with one or two colors such that its CNNG has the given edges. As part of this we observe that for a given set of input edges there are two situations that need special treatment. On the one hand, we have *local maxima* which are edges in the input such that both its left and right adjacent edge is in the input as well and shorter than the local maximum. On the other hand, there are *small gaps* which are missing edges between two consecutive input points such that this missing edge is not longer than both adjacent edges.

Situation	Parameters	Bound for color number	
No special edges	$k = 0, \ell = 0$	1	(Lemma 6.1)
No small gaps	$k \geq 0, \ell = 0$	$\lfloor \log(k+1) \rfloor + 1$	(Lemmas 7.1 and 7.4)
No local maxima	$k = 0, \ell \geq 0$	$\ell + 1$	(Lemmas 7.6 and 7.8)
No restrictions	$k \geq 0, \ell \geq 0$	$\ell + \lfloor \log(k+1) \rfloor + 1$	(Lemmas 7.9 and 7.10)

Table 1.2: An overview of the tight bounds for the number of colors in a colored nearest neighbor graphs for different situations. The bounds are relative to the number of local maxima k and the number of small gaps ℓ.

In Chapter 7 we look at whether there are upper and lower bounds on the number of color needed to solve our problem. Our first result in Section 7.1 is that for inputs which only have local maxima and no small gaps there is a tight logarithmic bound on the number of colors needed relative to the number of local maxima. For the number of local maxima itself, we know that there are at most half as many as the points in our input. In Section 7.2 we then see that allowing for small gaps there is a tight linear bound on the number of colors needed relative to the number of small gaps. Same as local maxima, there can be at most half as many small gaps as there are input points. Combining the previous results in Section 7.3 we see that the tight bound for inputs with both local maxima and small gaps is logarithmic in the number of local maxima and linear in the number of small gaps. See Table 1.2 for an overview of the achieved bounds.

Situation	Dynamic program running time …	
and parameters	…in terms of k and ℓ	…in terms of n
No small gaps ($k > 0, \ell = 0$)	$O\Big(\log^3 k \cdot (2k)^{1+\lfloor \log(k+1) \rfloor}\Big)$	$O\Big(\log^3 n \cdot n^{\log n}\Big)$
No local maxima ($k = 0, \ell > 0$)	$O\Big(\ell^{\ell+4}\Big)$	$O\Big(\big(\frac{n}{2}\big)^{\lfloor n/2 \rfloor + 3}\Big)$
No restrictions ($k, \ell > 0$)	$O\Big((\log k + \ell)^3 2^{\min(1+\lfloor \log(k+1) \rfloor + \ell, k)} (k+\ell)^{1+\lfloor \log(k+1) \rfloor + \ell}\Big)$	$O\Big(n^{\lfloor n/2 \rfloor + 3}\Big)$

Table 1.3: The running times of the dynamic program for colored nearest neighbor graphs in Chapter 8 for different parameters k and ℓ. The results are found in Theorem 8.1.

Finally, in Chapter 8 we design a dynamic program that is able to solve our problem for arbitrary number of colors. Its running time of

$$O\Big(\hat{c}^3 2^{\min(\hat{c},k)} m^{\hat{c}}\Big) \subseteq O\bigg(\hat{c}^3 2^{\min(\hat{c},\lfloor n/2 \rfloor)} \left(\frac{n}{2}\right)^{\hat{c}}\bigg)$$

depends on both on the number of special edges m and the number of colors $\hat{c}$. Thus, the upper bound on the running time depends on the number of local maxima k and the number of small gaps ℓ. See Table 1.3 for an overview over the running times depending on k and ℓ. We will also see that if we don't have local maxima, the running time of a naive approach that enumerates a (sufficiently large) subset of all possible color assignments is faster than our dynamic program. However, as long as $\ell \in o(k \log k)$ the dynamic program has better asymptotic behavior.

1.3 Publications

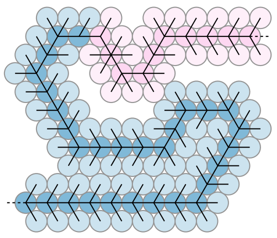
The results covered in the first part of this thesis are either novel or already appeared in the following publications. The results in the second part of this thesis are all novel and have not yet appeared in any publications.

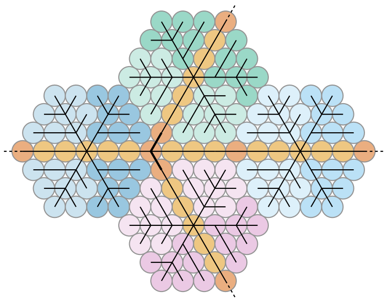
[CCN19] Man-Kwun Chiu, Jonas Cleve, and Martin Nöllenburg. "Recognizing Embedded Caterpillars with Weak Unit Disk Contact Representations Is NP-hard". In: *Proceedings of the 35th European Workshop on Computational Geometry (EuroCG)*. Utrecht, Netherlands, 2019

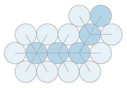
[Cle20] Jonas Cleve. "Weak Unit Disk Contact Representations for Graphs without Embedding". In: *Proceedings of the 36th European Workshop on Computational Geometry (EuroCG)*. Würzburg, Germany, 2020

I

Weak Unit Disk Contact Representations







Chapter 2

Preliminaries

In the following chapters we will investigate the problem of weak unit disk contact representations, as introduced in Section 1.2. In the problem we want to find a set of non-intersecting unit disks in the plane representing a given abstract graph. To do this we will first define what contact graphs and contact representations are in Section 2.1. Since we will look at the complexity of solving the problem for different graph classes, we briefly recap them in Section 2.2. In Sections 2.3 and 2.4 we look at some fundamental observations and definitions for the remaining chapters.

2.1 Contact Representations and Contact Graphs

Given a set of pairwise interior-disjoint unit disks in the plane, it is easy to find a graph representation such that each disk has a corresponding node and two nodes are adjacent if and only if their corresponding disks touch, i.e., their boundaries intersect. However, the inverse problem is generally not as easy: Given a graph, find a corresponding set of pairwise interior-disjoint unit disks in the plane. Such graphs and disk sets are formally defined as follows:

Definition 2.1. Let $G = (V, E)$ be a simple, undirected graph with $V = \{v_1, \dots, v_n\}$ and let $\mathscr{D} = \{D_1, \dots, D_n\}$ be a set of interior-disjoint unit disks in the plane such that D_i corresponds to v_i for all $1 \leq i \leq n$. We call $\mathscr{D}$ a *unit disk contact representation (UDCR)* of G if two disks touch, i.e., intersect in their boundaries, if and only if their corresponding nodes have an edge. A simple, undirected graph $G = (V, E)$ is a *unit disk contact graph (UDCG)* if it has a UDCR.

▷ unit disk contact representation (UDCR)
▷ unit disk contact graph (UDCG)

This model has been studied extensively [Bow+15; BK98; KNP15; KNP22], as seen in Section 1.1. Thus, in this work we study the following weaker version:

Definition 2.2. Let $G = (V, E)$ be a simple, undirected graph with $V = \{v_1, \dots, v_n\}$ and let $\mathscr{D} = \{D_1, \dots, D_n\}$ be a set of interior-disjoint unit disks in the plane such that D_i corresponds to v_i for all $1 \leq i \leq n$. We call $\mathscr{D}$ a *weak unit disk contact representation (weak UDCR)* of G if two disks touch (i.e., intersect in their boundaries) ↪

▷ weak unit disk contact representation (weak UDCR)
▷ weak unit disk contact graph (weak UDCG)

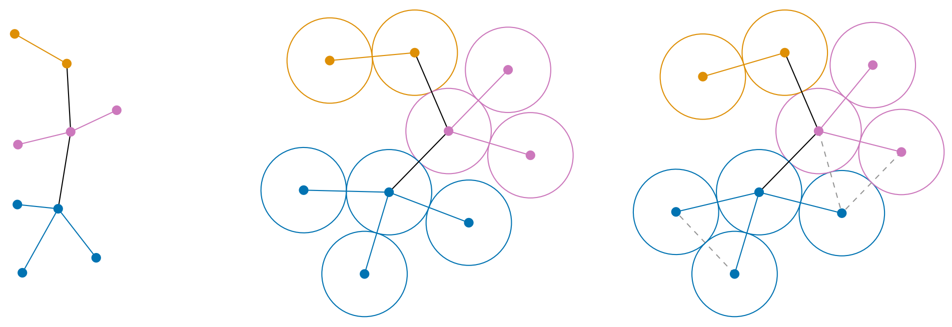


Figure 2.1: A graph (a) together with a possible corresponding UDCR (b) and a weak UDCR (c). The touching disks indicated by the dashed edges in (c) are only allowed in weak UDCRs.

if their corresponding nodes have an edge. A simple, undirected graph $G = (V, E)$ is a *weak unit disk contact graph (weak UDCG)* if it has a weak UDCR.

We observe that in Definition 2.1 we have an equivalence between edges and touching disks while in Definition 2.2 we only have an implication from edges to touching disks. Thus, in the weak setting, two disks are allowed to touch, even if no edge exists between the corresponding nodes. See Figure 2.1 for a graph together with a UDCR as well as a weak UDCR. In Figure 2.1c we can see that two disks are allowed to touch even though their corresponding nodes in the graph are not adjacent.

Let $\mathscr{D}$ be a weak UDCR for a graph $G = (V, E)$. Then for every node $v_i \in V$ with neighbors $v_{i_1}, \dots, v_{i_k}$ we have that $\mathscr{D}$ defines a cyclic order of $D_{i_1}, \dots, D_{i_k}$ around D_i. All cyclic orders for the nodes together form the *combinatorial embedding* Γ of G for the given weak UDCR. Note that, in general, there are infinitely many weak UDCRs which result in the same combinatorial embedding for G. If a graph is given together with a combinatorial embedding we can try to find a weak UDCR which results in this exact embedding:

▷ combinatorial embedding

Definition 2.3. Let $G = (V, E)$ be a simple, undirected graph with $V = \{v_1, \dots, v_n\}$ and let Γ be a combinatorial embedding of G. Let $\mathscr{D} = \{D_1, \dots, D_n\}$ be a set of interior-disjoint unit disks in the plane such that d_i corresponds to v_i for all $1 \le i \le n$. We call $\mathscr{D}$ an *weak embedded unit disk contact representation (weak embedded UDCR)* of G with respect to Γ if $\mathscr{D}$ is a weak UDCR of G and the combinatorial embedding of G for $\mathscr{D}$ equals Γ.

▷ weak embedded unit disk contact representation (weak embedded UDCR)

Since UDCRs are more constrained versions of weak UDCRs the following observation is obvious:

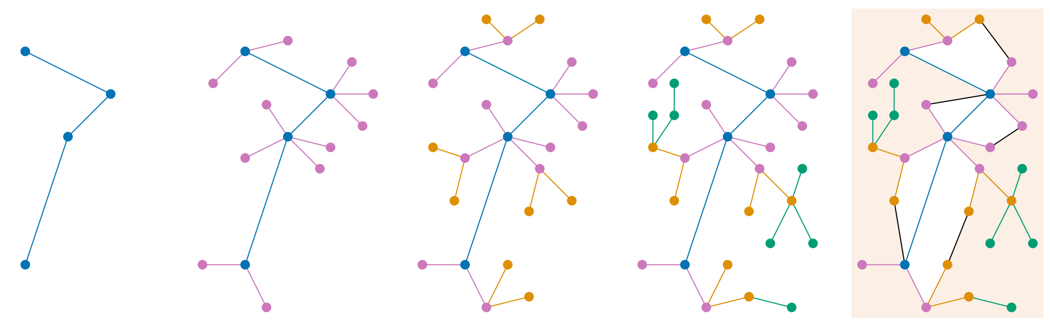


Figure 2.2: A progression of graphs that become more complex in each step. We start with a path (a) which is then the backbone for the caterpillar (b) and the lobster (c). In (d) every path we pick has nodes with distance more than two, but no cycles, making it a tree. Drawing (e) contains cycles, but we can see that every node is incident to the colored outer face.

Observation 2.1. *If a set of interior-disjoint unit disks $\mathcal{D}$ is a UDCR for a graph G, it also is a weak UDCR for G. Thus, if G is a UDCG, it is also a weak UDCG.* □

Corollary 2.1. *If a graph G has no weak UDCR, it also does not have a UDCR. Thus, if G is not a weak UDCG, it is also not a UDCG.* □

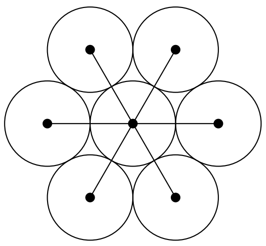
2.2 Important Graph Classes

We will look at NP-hardness or polynomial time algorithms for different graph classes. These graph classes will be defined here. Let $G = (V, E)$ be a simple, undirected, and connected graph. Then G is *planar* if it has a planar drawing, i.e., it can be drawn in the plane without intersecting edges; *outerplanar* if it has a planar drawing in which all nodes are incident to the outer face; a *tree* if its edges do not contain a cycle; a *path* if it is a tree and the degree of each node is at most 2; a *caterpillar* if it is a tree such that removing all leaves (i.e., nodes with degree 1) results in a path, which is called the caterpillar's *backbone*; and a *lobster* if it is a tree such that removing all leaves results in a caterpillar, whose backbone is the lobster's *backbone*. See Figure 2.2 for an example for each of the graph classes.

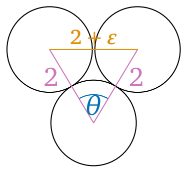
▷ planar
▷ outerplanar
▷ tree
▷ path
▷ caterpillar
▷ backbone
▷ lobster

2.3 Rigid Structures Using Tight Disk Packings

Klemz et al. [KNP22] show that it is NP-hard to decide whether an outerplanar graph has a UDCR or an embedded UDCR for a given embedding. They use internally-triangulated outerplanar graphs to force rigid structures to build gadgets for the



(a) The 6-star has only this unique weak UDCR.



(b) Looking at two neighboring leaf disks around the center disks. If $\varepsilon = 0$ then $\theta = \frac{\pi}{3}$ but setting $\varepsilon > 0$ implies $\theta > \frac{\pi}{3}$. Then six disks cannot be placed around the center disk.

Figure 2.3: Placing six disks around a center disk yields a tight packing in which each outer disk touches its two neighboring outer disks. It is not possible that they do not touch while maintaining six disks.

hardness proofs. For embedded UDCRs Bowen et al. [Bow+15] show NP-hardness even for trees using similar ideas. In all these proofs the structures cannot change significantly if we allow the disks of non-adjacent nodes to touch. Thus, it is already clear that the problem is NP-hard for outerplanar graphs and weak UDCRs as well as for trees and weak embedded UDCRs. As a result, the goal of this work will be to show NP-hardness for more constrained graph classes, namely trees and caterpillars. To do this, we will also need to force rigid structures via specially constructed graphs. We start by looking at what we mean by rigid.

Given a disk set $\mathscr{D}$ we translate it by a vector $\vec{v}$ and rotate it around a point p by an angle θ, by applying these transformations to each disk in $\mathscr{D}$ individually. We say that two sets of interior-disjoint unit disks $\mathscr{D}_1$ and $\mathscr{D}_2$ are *congruent* if and only if we can translate and rotate $\mathscr{D}_2$ obtaining $\mathscr{D}_2'$ such that $\mathscr{D}_1 = \mathscr{D}_2'$. This congruence relation is an equivalence relation which means that we can partition the set of all weak UDCRs of a given graph into equivalence classes. With this in mind, we can formulate our notion of rigidity.

▷ congruent

▷ rigid

Definition 2.4. A simple, undirected graph G is *rigid* if and only if it has a weak UDCR $\mathscr{D}$ and for all weak UDCRs $\mathscr{D}'$ of G it holds that $\mathscr{D}$ and $\mathscr{D}'$ are congruent.

A simple, undirected graph G together with a combinatorial embedding Γ is *rigid* if and only if it has a weak embedded UDCR $\mathscr{D}$ and for all weak embedded UDCRs $\mathscr{D}'$ of G it holds that $\mathscr{D}$ and $\mathscr{D}'$ are congruent. In other words, in both cases, G is *rigid* if the number of equivalence classes of its weak (embedded) UDCRs is exactly one.

Note that, since rigidity only compares the disk sets and not the underlying graph structure, the embedding of the weak UDCR is not important (unless, of course, we look for a weak embedded UDCR). With this in mind we can observe the following:

Observation 2.2. *The* 6*-star, i.e., the tree with one internal node and* 6 *leaves, is rigid.*

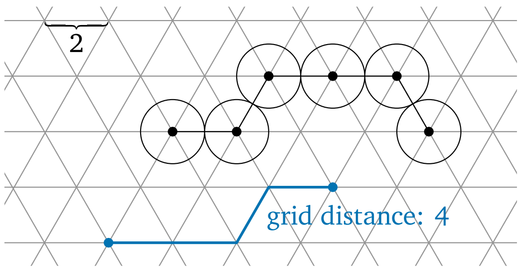


Figure 2.4: The triangular grid. In the top left corner we see that two adjacent grid points have distance 2. In the middle we have an example of a grid-restricted weak UDCR of a path with six nodes. Finally, on the bottom we have two grid points with grid distance 4.

Proof. Look at Figure 2.3a to see a weak UDCR of the graph together with its graph structure. In Figure 2.3b we can see that if two of the outer disks do not touch, then not all six disks can be placed around the center disks with distance exactly 2 between the centers of the outer and center disk without at least two outer disks intersecting. □

In the following we will use this simple rigid graph as a starting point for more complicated rigid structures, first without a given embedding, in Section 3.2, and then with a given embedding, in Section 4.2.

2.4 Triangular Grids

As we can see in Figure 2.3b in the densest possible packing of three disks, their centers form an equilateral triangle with side length 2. This single triangle can be extended to an infinite *triangular grid* such that two disks touch if and only if their centers are placed on neighboring grid points. Then the *grid size*, i.e., the distance between two neighboring grid point, is 2. We define the *grid distance* between two grid points as the number of edges on a shortest path between the points.

▷ triangular grid
▷ grid size
▷ grid distance

If we restrict ourselves to studying weak UDCRs and weak embedded UDCRs whose disks are all placed on triangular grid points instead of arbitrary locations, many tasks become easier. We thus define

Definition 2.5. A weak UDCR or weak embedded UDCR $\mathscr{D}$ is *grid-restricted* if it can be rotated and translated such that the centers of all disks are located on the triangular grid. A graph (together with an embedding) is *grid-representable* if it has a grid-restricted weak UDCR (or weak embedded UDCR).

▷ grid-restricted
▷ grid-representable

In Figure 2.4 we see an example of a grid-restricted weak UDCR as well as a depiction of the grid distance between two grid points.

Chapter 3
Graphs Without Embedding

In this chapter we focus on graphs where no specific combinatorial embedding is required. Thus, given a simple, undirected graph G, we want to find a weak UDCR $\mathscr{D}$ or decide that G has none; or in other words, decide whether G is a weak UDCG. The results presented in this chapter are twofold: In Section 3.1 we present a linear time algorithm to find weak UDCRs for caterpillars while in Section 3.2 we show that the problem becomes NP-hard for trees. These results were previously published in part and in short form in [Cle20].

3.1 A Linear Time Algorithm for Caterpillars

Our first contribution is an algorithm that, given a caterpillar, finds a weak UDCR in linear time or decides that no weak UDCR exists. Here some clarifying words are needed. In [Cle20] we claimed to have a condition that can be checked in linear time and which tells us whether a caterpillar has a weak UDCR. However, the claim that this condition is true if and only if a caterpillar has a weak UDCR is not correct—it is only a necessary but not a sufficient condition. We will show more details in Section 3.1.4. Before doing that, however, we present an alternate algorithm that runs in linear time. Note that in the meantime there has been some independent work on the same topic by Bhore et al. [Bho+21b]. The authors give a linear time algorithm to find a weak UDCR for grid-representable lobsters such that the positions of the disks of the backbone nodes are strictly x-monotone. They also claim that all grid-representable lobsters have a strictly x-monotone grid-restricted weak UDCR. However, in personal communication, Nöllenburg [Nöl24] confirmed that the proof for this claim was found to be flawed and the result is also not included in the PhD thesis of Terziadis [Ter24]. Their algorithm naturally works for grid-restricted, strictly x-monotone caterpillars. The algorithm presented here works for all caterpillars and is significantly simpler as it can use some properties of caterpillars.

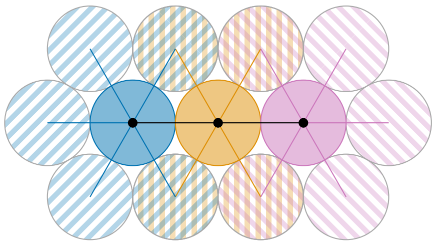


Figure 3.1: Positioning potential caterpillar disks on a triangular grid with the backbone disks on a horizontal line. We have three backbone nodes with different colors together with the possible placement positions of their leaf nodes. It is easy to see the five spots each disk at the end of the backbone has and that two consecutive backbone disks share two positions where they can place leaf disks.

3.1.1 Preliminary observation

Before looking at how to actually find a weak UDCR in linear time, we can make some observations about weak UDCRs:

Observation 3.1. *A k-star with $k > 6$ does not have a weak UDCR.*

Proof. We have seen in Observation 2.2 that the 6-star is rigid. Thus, no more disks can be added to the weak UDCR of the 6-star. □

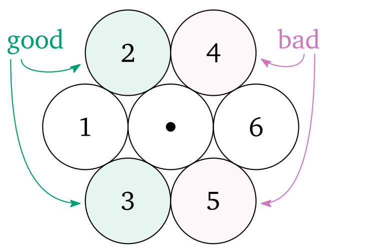
Corollary 3.1. *If a graph has a node with more than six neighbors, it does not have a weak UDCR.* □

3.1.2 The Linear Time Algorithm

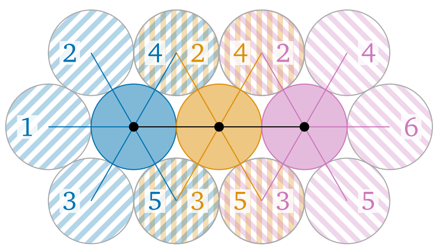
We now show our algorithm responsible for placing the caterpillar's disks. The idea is that all disks corresponding to the caterpillar's backbone are placed on a line and all disks corresponding to the leaves are placed on a triangular grid that coincides with the backbone disks. We call the resulting realization a *straight weak UDCR*. See Figure 3.1 for a depiction of three backbone disks and the possible leaf disks' positions. We can observe that the first and last backbone disks have five different spots for placing leaf disks while all other backbone disks have four such spots. Furthermore, two consecutive backbone disks share exactly two possible locations for their leaf disks.

▷ straight weak UDCR

The algorithm now works as follows. Given a caterpillar $G = (V, E)$, first determine its backbone consisting of nodes $b_1, b_2, \dots, b_k$ in this order. Then place all backbone disks on a horizontal line such that the disks of two consecutive nodes touch. Afterwards go through the backbone disks from left to right and try to place all their respective leaf disks according to the numbers shown in Figure 3.2. For every backbone disk its



(a) The positions on the triangular grid relative to the center disk are numbered according to this figure.



(b) We can see that adjacent backbone disks share two positions for leaf disks. These positions are 4 and 2 as well as 5 and 3.

Figure 3.2: The potential positions for disks corresponding to leaf nodes.

leaf disks are placed at positions $1, 2, 3, 4, 5, 6$ according to Figure 3.2a in this order if they are free. As we can see in Figure 3.2b position 1 is only free for the first backbone disk while position 6 is only free for the last backbone disk. If a leaf disk cannot be placed, it means that a backbone disk has more leaf disks than free positions around it. Then the algorithm returns that the given caterpillar does not have a weak UDCR.

Since positions 1 and 6 are special positions only relevant for the first and last backbone node we now focus on the remaining four positions. As we can see in Figure 3.2a, positions 2 and 3 are *good* positions, and 4 and 5 are *bad* positions. This is because, as we can see in Figure 3.2b, positions 2 and 3 do not overlap with the positions around the next backbone disk to the right. On the other hand, positions 4 and 5 occupy positions 2 and 3, respectively, of the next backbone disk to the right. The algorithm's placement order ensures that the leaf disks are only placed at bad positions if no good position is available.

▷ good position

▷ bad position

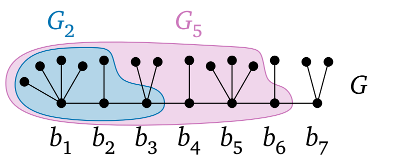
It is clear that the presented algorithm runs in linear time: Determining the backbone is done by finding all nodes with degree at least two. After placing the backbone on a horizontal line, every leaf is considered once and at most five positions are considered per leaf disk.

3.1.3 Correctness of the Linear Time Algorithm

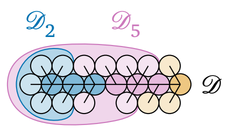
After seeing that the proposed algorithm is, indeed, a linear time algorithm we need to ensure that it is correct. It is obvious that if the algorithm successfully places all leaf disks, the result is a weak UDCR. Thus, it remains to show that if a caterpillar has a weak UDCR $\mathcal{D}$ the algorithm will find a weak UDCR $\tilde{\mathcal{D}}$. To this end we first define some properties and constructions that we use in the proof.

Definition 3.1. Given a caterpillar G with backbone nodes $b_1, \ldots, b_k$ in this order. The *i-restricted caterpillar* G_i of G, for $0 \leq i \leq k$ is the caterpillar obtained from G by removing all backbone nodes b_j with $j > i + 1$ and all leaves of b_{i+1}.

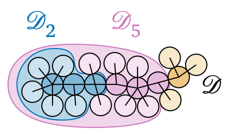
▷ *i*-restricted caterpillar



(a) A caterpillar with two restricted caterpillars highlighted.



(b) a weak UDCR as found by the algorithm.



(c) A possible arbitrary weak UDCR.

Figure 3.3: An example of a caterpillar and two possible weak UDCRs together with their 2- and 5-restricted variants highlighted.

See Figure 3.3a for an example caterpillar with two restricted parts highlighted. From the definition we observe that G_0 consists of only b_1 and $G_k = G$.

▷ i-restricted weak UDCR

Definition 3.2. Given a caterpillar G with a weak UDCR $\mathscr{D}$. The *i-restricted weak UDCR* $\mathscr{D}_i$ is $\mathscr{D}$ restricted to the disks corresponding to nodes in the i-restricted caterpillar G_i.

In Figures 3.3b and 3.3c we see two weak UDCRs for the caterpillar from Figure 3.3a and the two restricted weak UDCRs for the two restricted caterpillars, highlighted in the same colors. With the previous two definitions we can now look at how much space is available around each backbone disk to place its leaves and the next backbone disk. We define this space in terms of an angle around the disks in which other disks can be placed.

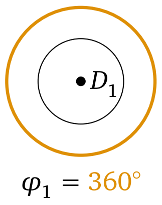
▷ free angle

We first define this *free angle* informally for a given weak UDCR $\mathscr{D}$. See Figure 3.4 for a depiction. For the first backbone node b_1 and its corresponding backbone disk D_1 we look at $\mathscr{D}_0 = \{D_1\}$. Then, we can place the leaf disks and D_2 at any position around D_1 and thus the free angle of D_1 is 360°, as seen in Figure 3.4a. For $\mathscr{D}_i$ with $1 \le i \le k$ we know that $\mathscr{D}_i$ contains D_i but not its leaves and not D_{i+1} (if it exists). Then the free angle of D_i is the angle of the largest cone towards the right such that any disk touching D_i placed completely inside the cone does not intersect with any disk from $\mathscr{D}_i$. In Figure 3.4b we can see the two dotted disks as the outermost positions for such disk placements. Thus, the cone contains the orange 180° part as well as the angles $\alpha_{i,l}$ and $\alpha_{i,r}$. These angles cannot be larger than 60° as a disk placed at a larger angle intersects D_{i-1}.

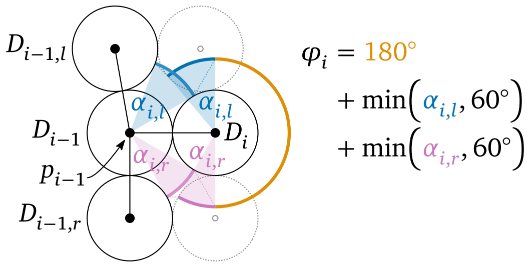
▷ free angle

Definition 3.3. Given a caterpillar G with backbone nodes $b_1, \dots, b_k$ in this order and a weak UDCR $\mathscr{D}$ of G. Let $D_i \in \mathscr{D}$ be the disk corresponding to b_i for all $1 \le i \le k$. We define the *free angle* φ_i of D_i with respect to the $(i-1)$-restricted weak UDCR $\mathscr{D}_{i-1}$ inductively. In the base case $\varphi_1 = 360°$ as $\mathscr{D}_0$ consists of only D_1, see Figure 3.4a.

For the inductive case look at Figure 3.4. Here we have indicated the two backbone disks D_{i-1} (with center p_{i-1}) and D_i as well as the two neighbors of D_{i-1} closest to D_i, namely $D_{i-1,l}$ and $D_{i-1,r}$. Let $\alpha_{i,l}$ be the angle between the closer tangents from p_{i-1} ↪



(a) The free angle φ_1 around D_1 is 360°.



(b) The free angle φ_i around D_i consists of 180° (orange) plus the (blue) angle $\alpha_{i,l}$ between D_i and $D_{i-1,l}$ and the (purple) angle $\alpha_{i,r}$ between $D_{i-1,r}$ and D_r; but at most 60° in both cases. The gray dotted disks indicate placement of two disks at the extreme positions around D_i.

Figure 3.4: The free angle around the disk D_1 of a 0-restricted weak UDCR and the disk D_i of a $(i-1)$-restricted weak UDCR. See Figure 3.5 why both blue and both purple angles are equal, respectively.

to $D_{i-1,l}$ and to D_i. If $D_{i-1,l}$ does not exist, let $\alpha_{i,l} = \infty$. Let $\alpha_{i,r}$ be the same for $D_{i-1,r}$ and D_i. Then $\varphi_i = 180° + \min\left(\alpha_{i,l}, 60°\right) + \min\left(\alpha_{i,r}, 60°\right)$.

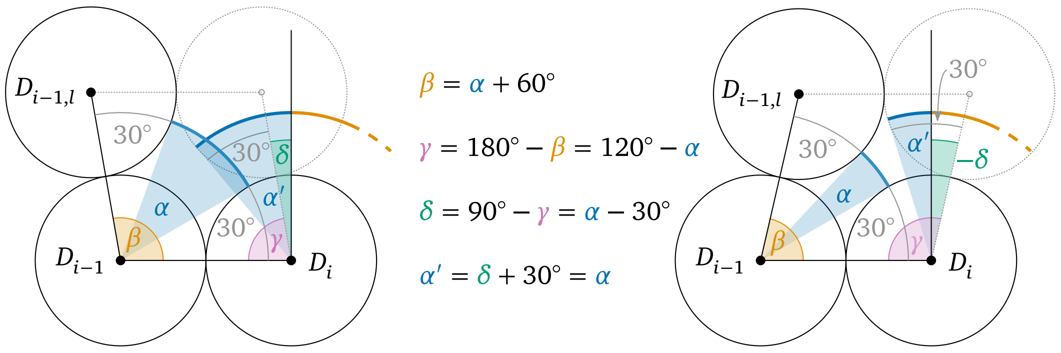
We need to show that this inductive definition equals the more informal definition from before. It is especially important to show that the two angles $\alpha_{i,l}$ and the two angles $\alpha_{i,r}$ in Figure 3.4b are actually the same:

Lemma 3.1. *Definition 3.3 of the free angle φ_i is exactly the angle of the largest cone towards the right such that any disk touching D_i placed completely inside the cone does not intersect with any disk from $\mathscr{D}_i$.*

Proof. We first show that the two angles denoted as $\alpha_{i,l}$ in Figure 3.4b are equal. For this, have a look at Figure 3.5 where we have a zoomed in view of the situation. The two angles are now denoted α and α'. The three centers of $D_{i-1,l}$, D_{i-1}, and D_i form a rhombus together with the center of the dotted disk. This is because the distances between the connected points are all 2.

If two disks touch, one disk occupies a sector of exactly 60° around the other disk. Thus, the angle β consists of α and half of each of the sectors containing $D_{i-1,l}$ and D_i. As a result, $\beta = \alpha + 60°$. Since we have a rhombus, we know that $\beta + \gamma = 180°$ which tells us that $\gamma = 120° - \alpha$. The right angle at the center of D_i is made up of γ and δ which means that $\delta = \alpha - 30°$. If $\gamma > 90°$ it follows that $\delta < 0$ which is not a problem as it only tells us that the center of the dotted disk is further to the right than the center of D_i. Finally, α' consists of δ and a sector containing exactly half of this dotted disk which makes it 30° large. Thus, $\alpha' = \alpha$ which tells us that indeed both angles denoted as $\alpha_{i,l}$ in Figure 3.4b are equal. By symmetry the same holds for $\alpha_{i,r}$.

↪



$$\beta = \alpha + 60°$$

$$\gamma = 180° - \beta = 120° - \alpha$$

$$\delta = 90° - \gamma = \alpha - 30°$$

$$\alpha' = \delta + 30° = \alpha$$

Figure 3.5: Two more detailed views of the upper part of Figure 3.4b to see that the two angles denoted by $\alpha_{i,l}$, here named α and α', are indeed equal. On the right side δ becomes negative if it is calculated as $90° - \gamma$.

If $D_{i-1,l}$ touches D_i then $\alpha_{i,l} = 0°$. In this situation we know that $D_{i-1,l}$'s center lies on the triangular grid whose grid points coincide with the centers of D_{i-1} and D_i. Then the furthest we can place a leaf disk D_i' of D_i is with a $60°$ angle between the line through the centers of D_{i-1} and D_i and the line through the centers of D_i and D_i'. Thus, the cone containing this disk starts at $90°$. Due to symmetry we then always have at $180°$ to place leaf disks around D_i. Together with the proof that the angles $\alpha_{i,l}$ and $\alpha_{i,r}$ in Figure 3.4b are respectively equal, we can conclude that Definition 3.3 correctly defines the largest cone towards the right such that any disk touching D_i placed completely inside the cone does not intersect with any disk from $\mathscr{D}_i$. □

With the correct definition of the free angle we can now have another look at the algorithm and observe that it always finds a weak UDCR for a caterpillar if it has one.

Lemma 3.2. *Let G be a caterpillar, $b_1, \dots, b_k$ its backbone nodes in this order, and $d_i = \deg(b_i)$ for all $1 \le i \le k$. If G has a weak UDCR $\mathscr{D}$ then the algorithm finds a weak UDCR $\tilde{\mathscr{D}}$ for G. For all $1 \le i \le k$ let φ_i be the free angle of b_i's disk with respect to $\mathscr{D}_{i-1}$ and $\tilde{\varphi}_i$ be the same with respect to $\tilde{\mathscr{D}}_{i-1}$; then $\varphi_i \le \tilde{\varphi}_i$.*

Proof. We first observe that $\tilde{\varphi}_i$ is a multiple of $60°$ for all $1 \le i \le k$. Let $\tilde{\alpha}_{i,l}, \tilde{\alpha}_{i,r}$ be the angles from Definition 3.3 when calculating $\tilde{\varphi}_i$. The algorithm places the disks on grid points which means that $\tilde{\alpha}_{i,l}, \tilde{\alpha}_{i,r} \in \{0°, 60°, 120°\}$ and thus $\tilde{\varphi}_i \in \{180°, 240°, 300°\}$. Furthermore, let $D_i \in \mathscr{D}$ be the disk corresponding to backbone node b_i for all $1 \le i \le k$. We now show the lemma by induction on the position $1 \le i \le k$ of the backbone node for which the algorithm places the leaf disks.

Induction base. For $i = 1$ we have that $\varphi_1 = \tilde{\varphi}_1 = 360°$ by definition and thus $\varphi_1 \le \tilde{\varphi}_1$ is true. We furthermore know that, $d_1 \le 6$ as otherwise $\mathscr{D}$ would not be a ↪

weak UDCR due to Observation 3.1. If $k = 1$ the algorithm places the d_1 leaf disks of D_1 at positions 1 up to d_1, and finishes successfully.

If $k \geq 2$, the algorithm has already placed D_2 at position 6 relative to D_1. Thus, it places the leaf disks of D_1 at positions 1 up to $d_1 - 1$. The result is $\tilde{\mathscr{D}}_1$. Now, given $\tilde{\mathscr{D}}_1$, we can obtain $\tilde{\varphi}_2$ and compare it with φ_2. For $i = 2$ we know that $\tilde{\alpha}_{2,l} \geq 60°$ if and only if $d_1 \leq 4$, otherwise $\tilde{\alpha}_{2,l} = 0°$. Similarly, $\tilde{\alpha}_{2,r} \geq 60°$ if and only if $d_1 \leq 5$, otherwise $\tilde{\alpha}_{2,r} = 0°$. It follows that $\tilde{\varphi}_2 = 180°$ if and only if $d_1 = 6$, $\tilde{\varphi}_2 = 240°$ if and only if $d_1 = 5$, and $\tilde{\varphi}_2 = 300°$ if and only if $d_1 \leq 4$. For φ_2 we know that d_1 disks use at least $d_1 \cdot 60°$ from φ_1 and there is at most $120°$ from φ_1 that can be used for φ_2. Thus, $\varphi_2 \leq 180° + \min(120°, 360° - d_1 \cdot 60°)$ which results in $\varphi_2 \leq 180°$ if $d_1 = 6$, $\varphi_2 \leq 240°$ if $d_1 = 5$, and $\varphi_2 \leq 300°$ if $d_1 \leq 4$. This shows that $\varphi_2 \leq \tilde{\varphi}_2$.

Induction step. For larger i we can now assume that $\varphi_i \leq \tilde{\varphi}_i$ is true before placing the leaf disks of D_i. Since $\mathscr{D}$ is a weak UDCR we know that $d_i - 1 \leq \left\lfloor \frac{\varphi_i}{60°} \right\rfloor$ as every neighbor of D_i uses at least $60°$; it is $d_i - 1$ since D_{i-1} is counted in d_i but is already placed. If $i = k$ the algorithm finishes after placing $d_i - 1$ leaf disks around D_i for which an angle of $\tilde{\varphi}_2$ available. This is sufficient, as $(d_i - 1) \cdot 60° \leq \left\lfloor \frac{\varphi_i}{60°} \right\rfloor \cdot 60° \leq \varphi_i \leq \tilde{\varphi}_i$.

If $i < k$, the algorithm will have already placed D_{i+1} at position 6, leaving $\tilde{\varphi}_i - 60°$ for the remaining $d_i - 2$ leaf disks. The algorithm can place these disks, since $(d_i - 2) \cdot 60° = (d_i - 1) \cdot 60° - 60° \leq \left\lfloor \frac{\varphi_i}{60°} \right\rfloor \cdot 60° - 60° \leq \left\lfloor \frac{\tilde{\varphi}_i}{60°} \right\rfloor \cdot 60° - 60° = \tilde{\varphi}_i - 60°$.

We now want to obtain $\tilde{\varphi}_{i+1}$. By construction, $\tilde{\alpha}_{i+1,r} \geq 60°$ if and only if $\tilde{\varphi}_i - (d_i - 1) \cdot 60° \geq 60°$, otherwise $\tilde{\alpha}_{i+1,r} = 0°$. Similarly, $\tilde{\alpha}_{i+1,l} \geq 60°$ if and only if $\tilde{\varphi}_i - (d_i - 1) \cdot 60° \geq 120°$, otherwise $\tilde{\alpha}_{i+1,l} = 0°$. This can then be combined into $\tilde{\varphi}_{i+1} = 180° + \min\left(\tilde{\alpha}_{i+1,l}, 60°\right) + \min\left(\tilde{\alpha}_{i+1,r}, 60°\right) = 180° + \min(120°, \tilde{\varphi}_i - (d_i - 1) \cdot 60°)$.

For φ_{i+1} we know that $d_i - 1$ disks are placed around D_i taking away $(d_i - 1) \cdot 60°$ from φ_i. From the remaining angle at most $120°$ can be part of φ_{i+1}. Thus, we can say that $\varphi_{i+1} \leq 180° + \min(120°, \varphi_i - (d_i - 1) \cdot 60°) \leq 180° + \min(120°, \tilde{\varphi}_i - (d_i - 1) \cdot 60°) = \tilde{\varphi}_{i+1}$. This completes the proof. □

We have just shown that the algorithm always produces a grid-restricted weak UDCR whenever there exists a (not necessarily grid-restricted) weak UDCR. It thus follows immediately, that

Corollary 3.2. *All caterpillars that have a weak UDCR have a grid-restricted weak UDCR and are thus grid-representable.* □

With the previous lemma we can now finalize this section with

Theorem 3.1. *Given a caterpillar G with n nodes, the algorithm in Section 3.1.2 runs in $\Theta(n)$ time and finds a weak UDCR for G if and only if G has a weak UDCR.*

Proof. At the end of Section 3.1.2 we analyzed the running time and saw that the algorithm does, indeed, run in linear time. It is obvious that, if the algorithm terminates successfully, the result is a weak UDCR: All backbone disks touch both their backbone neighbors' disks and the leaf disks are only placed at positions such that they touch their parent's disk. On the other hand, Lemma 3.2 shows that the algorithm always finds a weak UDCR if a given caterpillar has a weak UDCR. □

3.1.4 Incorrectness of the Previous Proof

After showing a correct linear time algorithm we now briefly show why the algorithm we presented in [Cle20] is not correct. Its correctness relied upon the following claim:

Wrong Claim 3.1 (Cf. [Cle20, Lemma 1]). *Let G be a caterpillar, $b_1, \dots, b_k$ its backbone nodes in order, and $d_i = \deg(b_i)$ for all $1 \leq i \leq k$. Then G has a weak UDCR if and only if for all $1 \leq \ell \leq k : \sum_{i=1}^{\ell} d_i \leq 4\ell + 2$.*

However, as we will see in the following example, this condition in the lemma is not a sufficient condition. Let G be our caterpillar with two backbone nodes b_1 and b_2 and seven leaf nodes, one leaf node for b_1 and six leaf nodes for b_2. Then $d_1 = 2$ (one leaf and b_2) and $d_2 = 7$ (six leaves and b_1). Checking the condition we have that for $\ell = 1$ it holds that $d_1 \leq 6$ and for $\ell = 2$ it holds that $d_1 + d_2 \leq 10$. However, as shown in Corollary 3.1 the graph cannot have a weak UDCR as b_2 has degree larger than six.

3.2 NP-hardness for Trees

In the previous section we showed that finding a weak UDCR for a caterpillar can be achieved in linear time. In this section we will show that the same task becomes NP-hard for trees. This is done by a reduction from Not-All-Equal 3SAT a variant of the 3SAT problem.

3.2.1 Not-All-Equal 3SAT and the Logic Engine Construction

▷ NAE3SAT
▷ valid assignment

Definition 3.4. *Not-All-Equal 3SAT (NAE3SAT)* is the problem of, given a 3SAT formula ϕ, finding a satisfying assignment for ϕ such that in every clause there is at least one false literal. Such an assignment is called *valid*.

The definition implies that for every valid assignment in every clause there is one true literal, one false literal, and one literal that can be either true or false. If a variable appears as both the positive and negative literal in the same clause we know that this clause is always satisfied, always has one true and one false literal, and can thus be removed. Henceforth, we can assume that formulas given to us do not contain such clauses. In 1978 Schaefer [Sch78] showed that NAE3SAT is an NP-hard problem. To

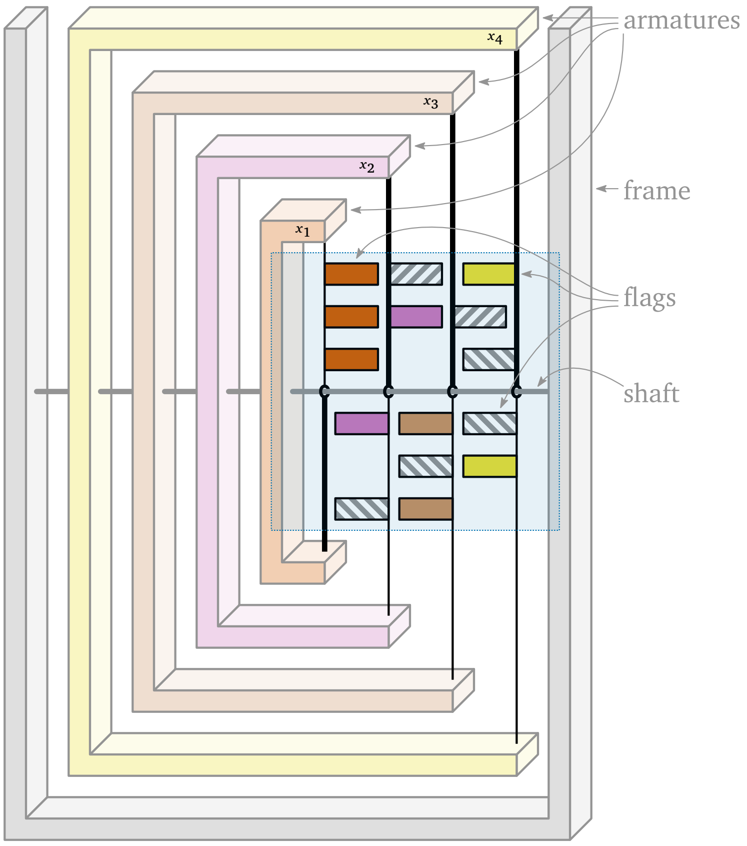


Figure 3.6: An example realization of the original logic engine by Eades and Whitesides [EW96] for the NAE3SAT formula $\phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_4) \land (x_1 \lor x_3 \lor \neg x_4)$. The outer U-shaped frame is fixed and the four C-shaped armatures each with a chain from top to bottom can be rotated around the gray horizontal shaft. The flags attached to the chains can be rotated around the chains. All important parts of the construction happen inside the highlighted area.

reduce NAE3SAT to a geometric realization problem such as finding a weak UDCR for a graph, we use a construction introduced by Bhatt and Cosmadakis [BC87], which was formalized by Eades and Whitesides [EW96] and has been called *logic engine* since then.

The original logic engine. We will first briefly describe this construction before going into more detail of our modified version of the logic engine. An example of a logic engine can be seen in Figure 3.6. It consists of a U-shaped fixed frame with a horizontal shaft in the middle. Around this shaft there are as many armatures as there are variables, four in our example. Each armature consists of a fixed C-shaped part with an additional vertical chain across the open side of the fixed part. The armatures can be rotated around the shaft into two possible positions that are in the same plane as the frame. At the chains of each armature, there are flags attached on as many levels on each side of the shaft as there are clauses in the NAE3SAT formula. In our example we have three clauses and thus three levels of flags on each side of the shaft. The flags can be rotated around the chains into two positions in the same plane as the frame and the armatures. However, certain positions are blocked by the innermost armature for x_1 and by the frame: The flags attached to the armature for x_1 must point towards the right while the flags attached to the armature for x_4 must point towards the left. Eades and Whitesides [EW96, Lemma 1] showed that there is a valid assignment for the formula if there exists a way to rotate the armatures and flags such that no two flags overlap.

A crucial observation of this construction, as seen in Figure 3.6, is that each armature is structurally a cycle and thus the logic engine contains several (nested) cycles. Since we want to show that finding a weak UDCR is NP-hard for trees, we need the structure of the logic engine to be tree-like as well. Fortunately, the important part of the construction is inside the highlighted area in Figure 3.6. We need each chain in either of the two possible positions and we need the flags that can be in up to two positions around their chains. Additionally, there needs to be something to the left preventing the leftmost flags from rotating to the left and the same on the right. In Figure 3.7 we can see such a construction which only consists of the important parts from Figure 3.6. We now describe this modified version of the logic engine. Since all rules for placing and moving the different parts resemble what happens in the original construction, NP-hardness of this construction will be immediate.

Our modified version of the logic engine. Given a NAE3SAT formula ϕ with variables $x_1, \dots, x_n$ and clauses $c_1, \dots, c_m$, the *logic engine* for ϕ is an orthogonal construction which can be realized in the plane without overlap if and only if there is a valid assignment for ϕ. See Figure 3.7 for the realization of a logic engine of a NAE3SAT formula with four variables and three clauses. The construction consists of a horizontal line segment, called the *shaft*, which is bounded to the left and right by vertical line segments of equal length, called the left and right *side*, which are attached to the shaft at their centers. In addition, n vertical line segments, called *flag poles*, with the same length as the sides are attached to the shaft at their centers. The leftmost flag pole is positioned very close to the left side and the rightmost flag pole is positioned very close to the right side. The other flag poles are positioned evenly spaced between the two outermost flag poles. From left to right the flag poles correspond to the variables x_1 to x_n in ϕ. Furthermore, the two parts of each flag pole on either side of the shaft correspond to the two possible choices for a variable and are called the *positive* and *negative* part. In Figure 3.7 the flag poles are thicker at the positive part and thinner at the negative part.

▷ logic engine

▷ shaft

▷ side

▷ flag pole

▷ positive / negative part

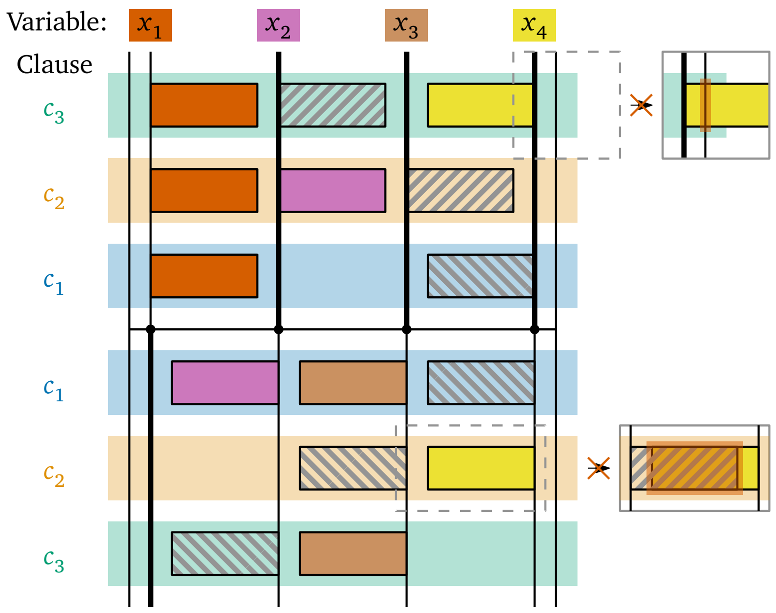


Figure 3.7: An example realization of a logic engine for the NAE3SAT formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee \neg x_4)$. The horizontal shaft is bounded by the two sides on the left and right. The four flag poles for the four variables with their thick positive and thin negative parts can be flipped around the shaft. The flags can be flipped to the left or right around the flag pole they are attached to. Since x_1's flag pole is flipped such that the negative part is on the top and all other flag poles are not flipped it means that the figure equals to setting $x_1 = 0$ and $x_2 = x_3 = x_4 = 1$ which is a valid assignment.
The upper box on the right shows that the outermost flag poles are forced to position their flags towards the center. The lower box shows that no two flags on the same level can be positioned towards each other.

The whole construction is then vertically subdivided into $2m$ levels of equal size: from the shaft upwards we have levels 1 to m and downwards we have levels -1 to $-m$. On both sides the levels correspond to clauses c_1 to c_m. The same subdivision into $2m$ levels is true for the flag poles, with levels 1 to m on the positive part and levels -1 to $-m$ on the negative part. That is, if the flag poles are not flipped (as explained later) the positions on the flag poles and the levels of the construction coincide. For every flag pole $i \in \{1, \dots, n\}$ and level $j \in \{1, \dots, m, -1, \dots, -m\}$ a *flag* may be attached to the ith flag pole. A flag is a rectangle of height slightly less than the height of a level and width slightly less than the width between two flag poles. For every variable $i \in \{1, \dots, n\}$ and clause $j \in \{1, \dots, m\}$ we attach flags to the ith flag pole according to the following rules. These rules are also depicted in Figure 3.8 and are the same rules as used by Eades and Whitesides [EW96]: ▷ flag

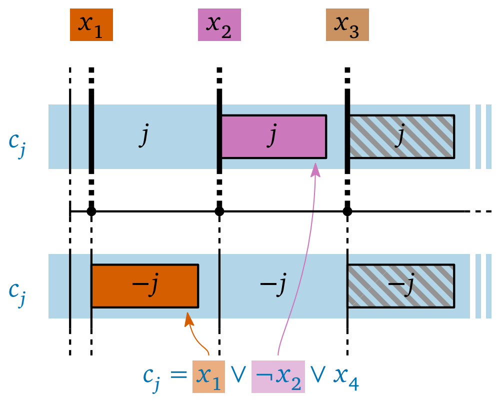


Figure 3.8: Depiction of the flag placing rules for clause $c_j = x_1 \vee \neg x_2 \vee x_4$ and three variables x_1, x_2, and x_3. 1. The positive literal x_1 forces a flag on $-j$. 2. The negative literal $\neg x_2$ forces a flag on j. 3. The non-appearing variable x_3 forces flags on both j and $-j$.

1. If x_i appears as the literal x_i in c_j, we add a flag to flag pole i on position $-j$, i.e., on the negative part.
2. If x_i appears as the literal $\neg x_i$ in c_j, we add a flag to flag pole i on position j, i.e., on the positive part.
3. If x_i does not appear in c_j, we add two flags to flag pole i, one on position j and one on position $-j$. These flags are the ones drawn with a hatching in Figure 3.7.

Given the logic engine, the task is now to draw it in the plane such that no parts of the drawing overlap. To be able to do this, each flag pole and each flag can be flipped individually. Each flag pole can be flipped vertically around the horizontal shaft and each flag can be flipped horizontally around its vertical flag pole. As discussed before and as we can conclude from the fact that Figure 3.7 is exactly the important part of Figure 3.6, our construction is functionally equivalent to the original logic engine. However, for the sake of completeness we provide the idea of the correctness of the reduction [EW96]. That is, a NAE3SAT formula has a valid assignment if and only if the corresponding (modified) logic engine construction can be drawn in the plane without overlap.

On each of the $2m$ levels there is enough space for $n-1$ flags, thus for every level there must be one flag pole which does not have a flag on this level. A variable x_i that does not appear in a clause c_j has a flag on the ith flag pole on positions j and $-j$. Thus, flipping the ith flag pole (i.e., changing x_i's truth value) has no impact on levels j and $-j$. It follows that if the ith flag pole does not have a flag on level j in a successful realization it can mean two things: Either x_i appears as x_i in c_j (a flag is placed only on position $-j$ of the flag pole) and the flag pole is not flipped (i.e., x_i = true) or x_i appears as $\neg x_i$ in c_j (a flag is placed only on position j of the flag pole) and the flag pole is flipped (i.e., x_i = false). In both cases the truth value for x_i also satisfies c_j. On level $-j$ the idea is the same with the only difference that the variable must be set to false if it appears as x_i in c_j and set to true if it appears as $\neg x_i$ in c_j. This means that the variable that does not have a flag on level $-j$ appears as a false literal in c_j. Thus, for every clause we have one true and one false literal. It also means that a realization directly corresponds to a valid assignment. If a flag pole is not flipped, i.e., the positive part is on the top, the corresponding variable is set to true. It is set to false if the flag pole is flipped, i.e., the negative part is on the top.

3.2.2 Rigid Hexagons as Basic Building Blocks

Our goal is now to model the structure of the logic engine by a weak UDCR of a tree. We do this step by step, starting with very basic rigid structures: As discussed in Section 2.3, one such simple rigid structure is obtained by tightly packing unit disks as a weak UDCR of a 6-star. The disk centers of this weak UDCR coincide with the grid points of the triangular grid, as defined in Section 2.4. The main rigid structure we use for the construction is a rooted tree whose weak UDCR is a hexagon, as already suggested by the section title. This tree and its weak UDCR can come in arbitrary sizes. For the hexagonal weak UDCRs we will use the term *radius* for the largest grid distance between the disk in the center and any other disk of the weak UDCR. In the corresponding tree, this will be equivalent to the height of the tree. However, we will refer to it as the *radius* of the tree as well.

In Figure 3.9 we can see different examples of trees with various radii r and in Figure 3.10 we see possible weak UDCRs for those trees. More formally, we define those trees as follows:

Definition 3.5. The *rigid tree* $R(r)$ with radius $r \in \mathbb{N}$ is a rooted tree defined as follows. For $r = 0$ it consists of a single node, while for $r \geq 1$ it is a node with six children. From those six children there are exactly two of each of the following subtrees: $P(r)$, $A(r)$, and $B(r)$. These subtrees are defined as follows.

▷ rigid tree

- $P(r)$ is a path consisting of r nodes. These are the orange parts in Figures 3.9 and 3.10.
- $A(r)$ is a single node for $r = 1$. Otherwise, it consists of a root node with three children: one single node, one subtree $P(r-1)$, and one subtree $A(r-1)$. These are the purple parts in Figures 3.9 and 3.10.
- $B(r)$ is a single node for $r = 1$. Otherwise, it consists of a root node with two children: one subtree $P(r-1)$ and one subtree $B'(r-1)$. These are the light blue parts in Figures 3.9 and 3.10.

↪

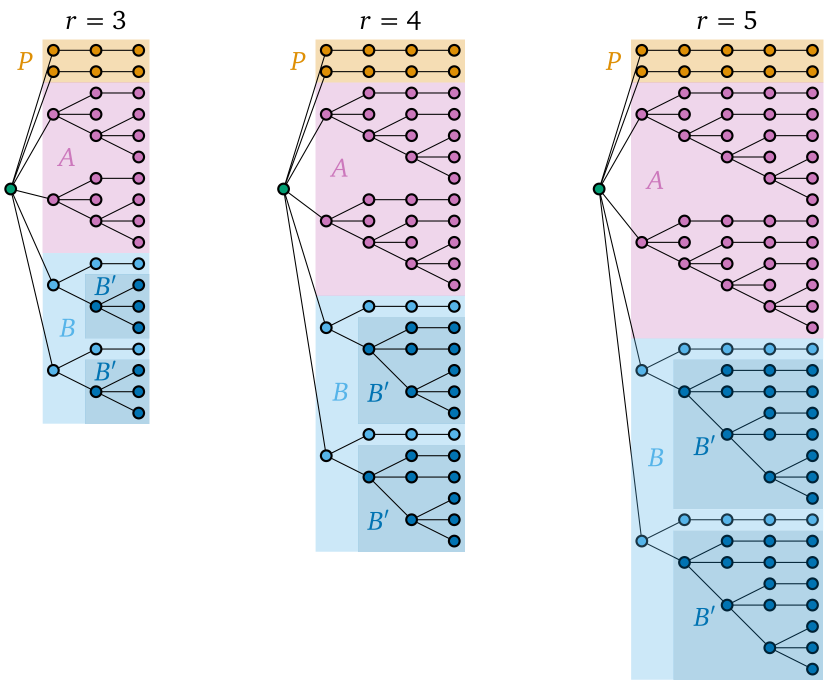


Figure 3.9: Three rigid trees with radii $r = 3, 4, 5$. Here, radius means the number of edges on a longest simple root-leaf-path. The corresponding weak UDCRs are found in Figure 3.10.

- $B'(r)$ is a single node for $r = 1$. Otherwise, it consists of a root node with three children: two subtrees $P(r-1)$ and one subtree $B'(r-1)$. These are the dark blue parts in Figures 3.9 and 3.10.

In addition, we define two functions to count the nodes in a tree $T(r)$ where T can be one of R, P, A, B, and B': We define $N_T(r)$ as the total number of nodes in $T(r)$ and $N_T^i(r)$ as the number of nodes in $T(r)$ with distance $i-1$ from the root[a].

[a]This means that $N_T^1(r) = 1$ for all $r \geq 1$ and all T as it only counts the root node.

To show that these trees are actually rigid, as the name suggests, we need to know how many nodes there are in the different tree and subtree types; in total as well as those with a certain distance to the root node. For this we make

Observation 3.2. *The following equalities regarding the number of nodes of the different tree types are all true:*

- $N_P^i(r) = 1$ *for all* $1 \leq i \leq r$ *and* $N_P(r) = r$.
- $N_A^i(r) = i + 1$ *for all* $2 \leq i \leq r$, $N_A^1(r) = 1$, *and* $N_A(r) = \frac{r^2+3r-2}{2}$.
- $N_{B'}^i(r) = 2i - 1$ *for all* $1 \leq i \leq r$ *and* $N_{B'}(r) = r^2$.

↪

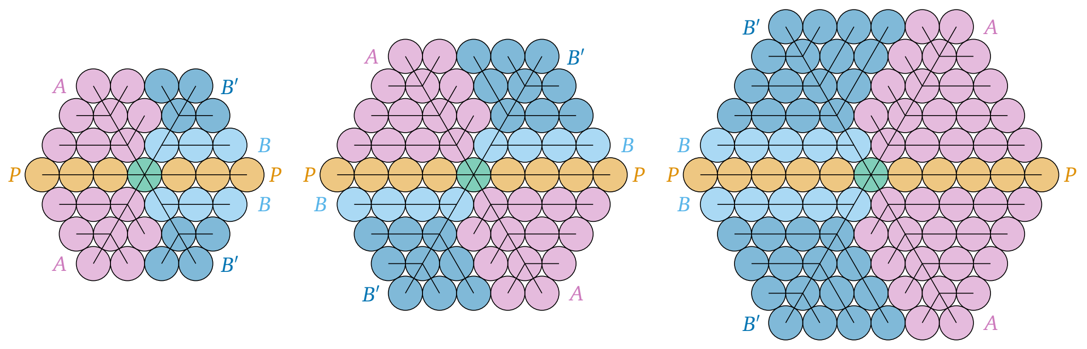


Figure 3.10: Possible weak UDCRs of the three rigid trees with radii $r = 3, 4, 5$ as found in Figure 3.9. The blue and purple parts on the same side of the orange disks can be swapped arbitrarily.

- $N^i_B(r) = 2i - 2$ *for all* $2 \le i \le r$, $N^1_B(r) = 1$, *and* $N_B(r) = r^2 - r + 1$.
- $N^i_R(r) = 6(i-1)$ *for all* $2 \le i \le r$, $N^1_R(r) = 1$, *and* $N_R(r) = 3r^2 + 3r + 1$.

Proof. We already observed before that $N^1_T(r) = 1$ for all $r \ge 1$ and T, since this counts exactly the one root node. By definition, $P(r)$ consists of r nodes and for every i there is exactly one node since $P(r)$ is a path.

Looking at $A(r)$'s definition we see that for $i = 2$ we have

$$N^2_A(r) = 1 + N^1_P(r-1) + N^1_A(r-1) = 1 + 1 + 1 = 2 + 1 = i + 1,$$

and for larger distances i with $2 < i \le r$ we can use induction to see that

$$N^i_A(r) = N^{i-1}_P(r-1) + N^{i-1}_A(r-1) = 1 + ((i-1) + 1) = i + 1.$$

Adding $N^i_A(r)$ up for all $1 \le i \le r$ gives us

$$\begin{aligned} N_A(r) = N^1_A(r) + \sum_{i=2}^{r} N^i_A(r) &= 1 + \sum_{i=2}^{r}(i+1) = r - 1 + \sum_{i=1}^{r} i \\ &= r - 1 + \frac{r^2 + r}{2} = \frac{r^2 + 3r - 2}{2}. \end{aligned}$$

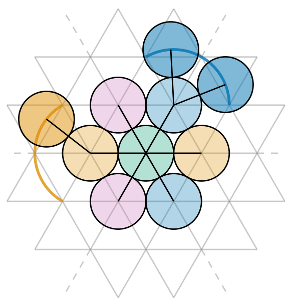
With $B'(r)$'s structure we can use induction (for which the base case is already shown) and show that

$$N^i_{B'}(r) = 2N^{i-1}_P(r-1) + N^{i-1}_{B'}(r-1) = 2 + (2(i-1) - 1) = 2i - 1$$

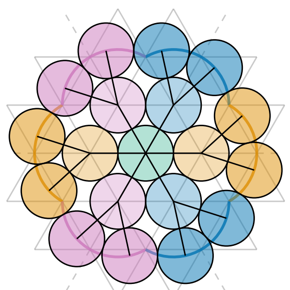
for all $2 \le i \le r$ and thus, after adding it all up we obtain

$$N_{B'}(r) = \sum_{i=1}^{r} N^i_{B'}(r) = \sum_{i=1}^{r}(2i-1) = -r + 2\sum_{i=1}^{r} i = -r + r^2 + r = r^2.$$

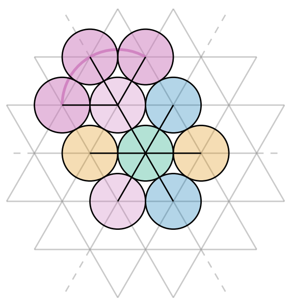
↪



(a) One or two children on their own do not need to be placed on the grid.



(b) Even if all six nodes have two children, they can be placed outside the grid.



(c) One node with three children will force all of them onto the grid.

Figure 3.11: Depending on the number of children of the 6-star, the corresponding disks are not necessarily placed on the grid. The constraint is that the disk centers are placed on the colored arcs.

For $B(r)$ it follows that for $2 \le i \le r$ we have

$$N_B^i(r) = N_P^{i-1}(r-1) + N_{B'}^{i-1}(r-1) = 1 + (2(i-1)-1) = 2i-2$$

and adding it all up we get

$$\begin{aligned} N_B(r) &= N_B^1(r) + \sum_{i=2}^{r} N_B^i(r) = 1 + \sum_{i=2}^{r}(2i-2) = 1 + \sum_{i=1}^{r}(2i-2) \\ &= 1 - 2r + 2\sum_{i=1}^{r} i = 1 - 2r + r^2 + r = r^2 - r + 1. \end{aligned}$$

Finally, looking at $R(r)$ we see that for $2 \le i \le r+1$ we have

$$\begin{aligned} N_R^i(r) &= 2N_P^{i-1}(r) + 2N_A^{i-1}(r) + 2N_B^{i-1}(r) \\ &= 2 + 2((i-1)+1) + 2(2(i-1)-2) = 2 + 2i + 4i - 8 = 6i - 6 = 6(i-1). \end{aligned}$$

After adding everything up for all $1 \le i \le r+1$ we obtain

$$N_R(r) = 1 + \sum_{i=2}^{r+1} 6(i-1) = 1 + \sum_{i=1}^{r} 6i = 1 + 3r(r+1) = 3r^2 + 3r + 1. \qquad \square$$

We are now able to see that rigid trees are indeed, as the name suggests, rigid graphs as defined in Definition 2.4 and that their weak UDCRs form the shape of a hexagon.

Lemma 3.3. *All rigid trees $R(r)$ for all $r \in \mathbb{N}$ are rigid. Furthermore, the weak UDCRs of $R(r)$ for all $r \in \mathbb{N}$ are grid-restricted. For $r \ge 1$ the shape of the weak UDCRs (looking at the convex hull of all disk centers) is a regular hexagon with side length $2r$ and all grid points inside the hexagon or on its boundary are occupied by disk centers.*

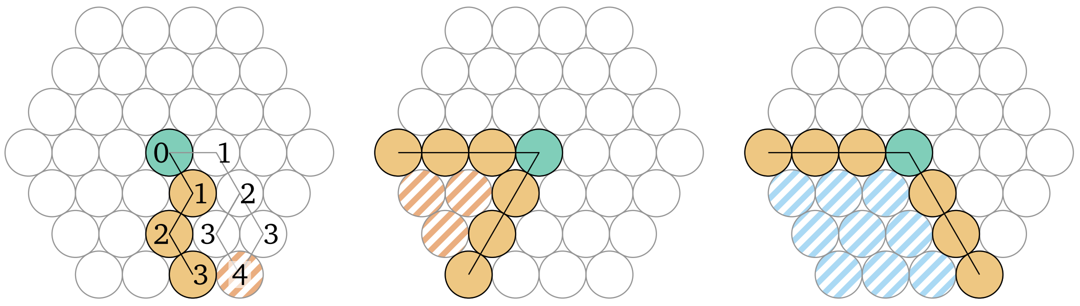
Proof. We show this by induction on the radius r. For $r = 0$ it holds trivially as we have just one disk. For $r = 1$ we have a 6-star which, as seen in Observation 2.2, is rigid. Since its weak UDCR is a tight packing of seven disks, as seen in Figure 2.3a, it is grid-restricted. It is also true that the convex hull of all disk centers is a regular hexagon with side length 2. This hexagon contains exactly seven grid points: the six on the boundary and the one in the center. Thus, all seven grid points covered by the hexagon are occupied by disk centers.

We now assume that the lemma holds for $r - 1$. Let $\mathscr{D}'$ be a weak UDCR of $R(r-1)$. We try to construct $\mathscr{D}$, a weak UDCR of $R(r)$. First, we look at the nodes with distance to root $r - 1$ that have exactly one or two children. Note that these children are leaves in $R(r)$ and do not exist in $R(r-1)$. Placing their disks individually does not enforce any specific location; the only constraint is that the disk centers must be placed on a circular arc, see Figure 3.11a. Even if all nodes with distance $r - 1$ to the root had exactly two children, their placement would not be unique: Placing one child forces the position of all other children, but the first child has a movement freedom of 60°. See Figure 3.11b for a depiction of the situation where we can also see that the resulting shape would not be a hexagon but a dodecagon. We also see that the disk centers are not necessarily on the grid.

However, at least one node with distance $r - 1$ to the root has three children, a node in the subtree of type A. Looking at Figure 3.11c, we see that around the corresponding disk, all children's disk centers must be placed on a circular arc of 120°. The two sibling disks and the parent disk already occupy 180° leaving another 180° for the remaining three disks. Since we talk about the children's disk centers, we remove 30° on both sides, leaving the 120° arc. The disk centers need to have distance at least 60° between them, leaving no other choice than placing two at the extreme points and one in the middle.

There are $N_R^{r+1}(r) = 6r$ nodes with distance r from the root, exactly those added to $R(r-1)$ to obtain $R(r)$. Additionally, given any fixed point in a triangular grid, there are $6s$ grid points with grid distance s from it. Thus, if the three leaf disk in the subtree of type A are forced to be placed on the grid, all other leaf disks in $R(r)$ are forced to be placed on the grid as well. It follows that all weak UDCRs for $R(r)$ place all their disks on the grid points with distance at most r on the triangular grid, centered in the root disk. Thus, all weak UDCRs of $R(r)$ are equal up to rotation and translation, and henceforth, $R(r)$ is rigid. In addition, taking the convex hull of all grid points with distance exactly r from the root disk's center, i.e., the outermost occupied grid points, forms a regular hexagon with side length $2r$.

By induction, in any weak UDCR of $R(r-1)$ all grid points inside or on the boundary of the hexagon with side length $2r - 2$ are occupied by disk centers. We then only need to confirm that for any weak UDCR of $R(r)$ the additional disks occupy all grid points of the larger hexagon's boundary. This is easy to see, since we already observed



(a) Not placing all disks of a subtree on a line through the center results in the marked position being left empty.

(b) Forming a 60° angle between two subtrees leaves a non-empty region where no disk can be placed.

(c) Forming a 120° angle between the two subtrees leaves an area which cannot be filled exactly by a subtree of type A or B.

Figure 3.12: Illegal placements of subtrees of type P in a weak UDCR for a chainable hexagon tree with $r = 3$.

that there are exactly as many additional disks in $R(r)$ compared to $R(r-1)$ as there are grid points with distance r from the center disk's grid point. Thus, the new disks must occupy all grid points on the boundary of the larger hexagon.

This concludes the induction step and thus the proof. □

In Figure 3.10 we see one possible weak UDCR for each of the three rigid trees with radii 3, 4, and 5. We can see that in all three realizations disks of nodes in the subtrees of type P are on a line. This is, indeed, true for all weak UDCRs of rigid trees with radius $r \geq 3$. We call them *chainable hexagon trees* and prove our claim in the following

▷ chainable hexagon tree

Lemma 3.4. *In all weak UDCRs of chainable hexagon trees, i.e., all rigid trees with radius $r \geq 3$, the disks corresponding to the two leaf nodes of the subtrees of type P are in opposite corners of the resulting hexagon.*

Proof. We are given $R(r)$ for some $r \geq 3$ together with a weak UDCR $\mathscr{D}$. Remember that by the previous lemma all disks are placed on a triangular grid and inside a regular hexagon and all positions inside the regular hexagon are occupied by disks. Let $D_R \in \mathscr{D}$ be the disk corresponding to the root node of $R(r)$.

We first show that in every weak UDCR of $R(r)$ each subtree of type P places its disk centers on a line through the center of D_R. Assume that there is a weak UDCR where the disk centers of a subtree P' of type P are not on a line together with the root disk. Let $D_{P'} \in \mathscr{D}$ be the disk corresponding to the root node of P' and let it, w.l.o.g., be located to the lower right of D_R. See Figure 3.12a for an example of such a placement of P' for $r = 3$. Now we look at the position on the triangular grid where ↪

the leaf node of P' would be placed if it were on a line through the center of D_R. This position is marked in Figure 3.12a and has a 4 in it. The position has grid distance r from the center of D_R and there is exactly one shortest path to it: going straight to the lower right without any bends. However, to place a disk on the marked position, there needs to be a leaf in $R(r)$ with distance at least $r+1$ to the root: The shortest path towards the position cannot be taken as it is blocked by $D_{P'}$. Figure 3.12a shows two possible paths, both of which have length $r+1$. Since no node with distance to the root more than r exists in $R(r)$, there cannot be a disk at the marked position. However, as observed before, all positions in the hexagon must be occupied for a weak UDCR of $R(r)$. Thus, P' must be placed on a line through the center of D_R.

We now show that if each subtree of type P places their disks on a line through the center of D_R, then they are placed on the same line. Assume, without loss of generality, that one subtree places its disks towards the left. If the second places its disks towards the right, the statement is already true. Thus, four remaining directions, which we can split into two cases due to symmetry, need to be checked:

1. The second subtree of type P places its disks on a line towards the lower left (or symmetrically, the upper left). This situation is depicted in Figure 3.12b. Here, the hatched disks are positions where no disk can be placed. For all $r \geq 3$ there are at least three such positions and thus, this placement is not possible.

2. The second subtree of type P places its disks on a line towards the lower right (or symmetrically, the upper right). A depiction of this situation is found in Figure 3.12c. Here, there are exactly $\sum_{i=1}^{r} 2i-1 = r^2$ hatched positions. Those positions need to be filled by exactly one subtree of type A or B. However, Observation 3.2 tells us that $A(r)$ has $N_A(r) = \frac{r^2+3r-2}{2}$ and $B(r)$ has $N_B(r) = r^2 - r + 1$ nodes which for $r \geq 3$ is both less than r^2. Thus, this placement of the second subtree of type P is impossible, as well.

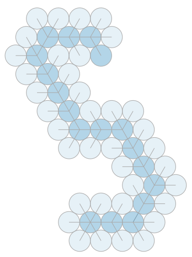
We can conclude that both subtrees of type P are placed on the same line through D_R and thus the two leaves of those subtrees are on opposite corners of the hexagon. □

3.2.3 Combining Hexagons Into Line Segments

We can now construct chainable hexagon trees with two specific leaf nodes on opposite corners of their weak UDCRs. Taking two such trees and joining them together by merging one such leaf node of each tree into a new node results in a new tree which is supposed to represent line segments. We define such trees inductively.

Definition 3.6. A *line segment tree* is an inductively defined rigid tree with two specific *connector nodes*. Any single chainable hexagon tree $R(r)$ is a line segment tree with the two connector nodes being the two leaf nodes of the two subtrees of type P. We ↪

▷ line segment tree
▷ connector node
▷ spine
▷ length



combine two line segment trees T_1 and T_2 with connector nodes v_1 and v'_1 as well as v_2 and v'_2 by merging v'_1 and v_2 into one node. The result is a line segment tree T with connector nodes v_1 and v'_2.

The *spine* of a line segment tree is the simple path from one connector node to the other and the *length* of a line segment tree is the number of nodes on the spine.

See Figure 3.13 for an example line segment tree consisting of three chainable hexagon trees. When combining two line segment trees we lose one node on the spine as seen in

Observation 3.3. *The length of a line segment tree is $2r+1$ in the base case and $\ell_1+\ell_2-1$ when combining two line segment trees with lengths ℓ_1 and ℓ_2.*

Proof. The spine of a chainable hexagon tree $R(r)$ (the base case) consists of the two paths $P(r)$, each with r nodes, and the root node. This results in $2r + 1$ nodes on the spine. When combining two line segment trees with lengths ℓ_1 and ℓ_2, two individual nodes on each spine are merged into one new node. All other nodes on the spine stay the same. Thus, $\ell_1 - 1 + \ell_2 - 1 + 1 = \ell_1 + \ell_2 - 1$ nodes are on the spine of the resulting line segment tree. □

With this observation we conclude that we can construct a line segment tree for nearly any length:

Corollary 3.3. *Given an* odd *natural number $\ell \geq 7$, there is a line segment tree with length ℓ which is constructed by combining at most one chainable hexagon tree $R(4)$ or $R(5)$ and a finite number of chainable hexagon trees $R(3)$.*

Proof. For $r \in \{3, 4, 5\}$ we have the following: For any $\ell = (2r + 1) + 6k$ with $k \in \mathbb{N}$ we can combine one chainable hexagon tree $R(r)$ with k times $R(3)$. This results in length $2r + 1$ for $R(r)$ if $k = 0$ and adds 6 for every time we add another $R(3)$.

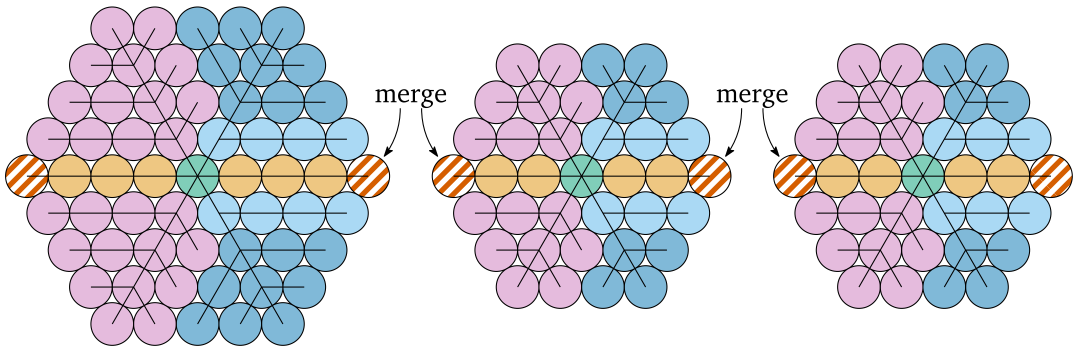
For any odd $\ell \geq 7$ we have that $\ell \bmod 6 \in \{1, 3, 5\}$. If $\ell \bmod 6 = 1$ we can write $\ell = (6+1)+6k = (2r+1)+6k$ for $r = 3$. Similarly, we obtain $r = 4$ for $\ell \bmod 6 = 3$ and $r = 5$ for $\ell \bmod 6 = 5$. □

We now show that line segment trees behave like chainable hexagon trees:

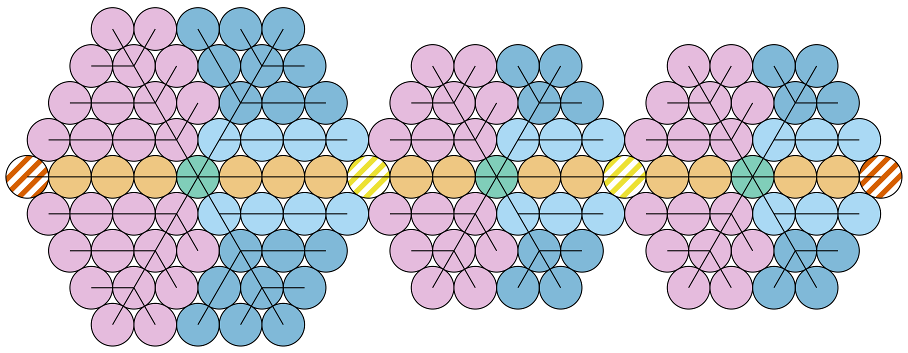
Lemma 3.5. *Let T be a line segment tree with connector nodes v and v'. Then T is rigid and for all weak UDCRs the disks corresponding to the spine nodes are placed on a line.*

Proof. We show this by induction on the number of combining steps to inductively construct T. We also show that in all weak UDCR both connector node disks have three touching disks each, one spine disk and one on either side of the line on which the spine disks are placed.

↪



(a) Three weak UDCRs of the chainable hexagon trees with their connector disks (the disks corresponding to their connector nodes) marked. The indicated connector disks will be merged into one disk.

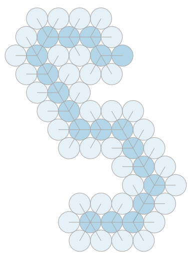


(b) After merging the connector disks we see the weak UDCR of the line segment tree.

Figure 3.13: The weak UDCRs of how to construct a line segment tree with length 21. The underlying tree consists of one $R(4)$ and two $R(3)$.

In the base case T is a chainable hexagon tree and from Lemma 3.4 our statement follows directly. We also observe that in every weak UDCR the disks corresponding to the two connector nodes are placed in the corners of the hexagon. Thus, it has three touching disks, one of which corresponds to the parent node and the two other are on either side of the line on which the spine disks are placed.

In the induction step we assume that we are given two line segment trees T_1 and T_2 for which the induction hypothesis holds. By combining one connector node from each line segment tree, in every weak UDCR the corresponding disk will have exactly six touching disks. Six disks around a center disk force rigidity of those six disks (cf. Observation 2.2). Thus, no arbitrary movement around the connection disk is possible. It remains to show that the connection forces all spine disks on one line which will conclude the rigidity of the combined construction. Due to the fact that both in T_1 and T_2 the connector node disk has exactly one touching disk on either side of the line ↪



through the spine disks, there will be exactly two non-spine disks between the two spine disks around the combined node's disk. Thus, the two spine disks together with the combined node's disk are on a line and, henceforth, all spine disks are on the same line. The two connector nodes of the resulting line segment tree did not change while combining T_1 and T_2 and thus all properties remain as before. □

3.2.4 A Branching and Flipping Gadget

Our end goal is to simulate the logic engine presented in Section 3.2.1. To do this, we need to be able to model more than just line segments. Thankfully, the only additional construction we need is a four-way intersection which is fixed along one axis and allows the two side branches on the other axis to be exchanged. This will allow us to simulate the flag poles as well as the flags. For the flag poles one side is the positive and the other the negative part and exactly one of them needs to be placed on the top and the other one on the bottom while the shaft runs horizontally. For the flags one side is the actual flag and the other is just an empty side branch with nothing attached. They both need to be flipped left and right while the flag pole runs vertically.

We now define the gadget to model the four-way intersection. A depiction of possible weak UDCRs can be seen in Figure 3.14.

▷ branching gadget

Definition 3.7. The *branching gadget* $B(r, r')$ for $r, r' \in \mathbb{N}$ with $r, r' \geq 3$ is a tree that is constructed by combining four chainable hexagon trees and a path. Let $P' = P(3)$ be the path and $R_1 = R(r)$, $R_2 = R_3 = R(3)$, and $R_4 = R(r')$ be the chainable hexagon trees. Similarly to Definition 3.6, we call the leaf nodes of the subtrees of type P in the chainable hexagon trees *connector nodes*. We add three children to one connector node of R_1: one connector node of R_2, one of R_3 and one leaf of P'. Then add an edge between the remaining leaf of P' and one connector node of R_4.

We can directly see that the branching gadget has four connectors To ensure that the branching gadget fulfills the requirements that it is fixed along one axis and allows the two side branches to be exchanged we show the following

Lemma 3.6. *For $r, r' \in \mathbb{N}$ with $r, r' \geq 3$ the branching gadget $B(r, r')$ is rigid. Let v be the connector node of R_1 to which R_2, R_3, and P' are added as children. In any weak UDCR of $B(r, r')$, consider R_1 to be fixed. Then, the positions of P' and R_4 are fixed relative to D_v, the disk corresponding to v. Furthermore, there are exactly two fixed positions relative to D_v that can be occupied by R_2 and R_3.*

Proof. We know from Lemma 3.3 that R_1, R_2, R_3, and R_4 are rigid. Additionally, the disk corresponding to R_1's connector node at which the three children are attached has six touching disks. Hence, the three positions of the three disks of the children's connector nodes are fixed on the triangular grid. Therefore, it suffices to see in which ↪

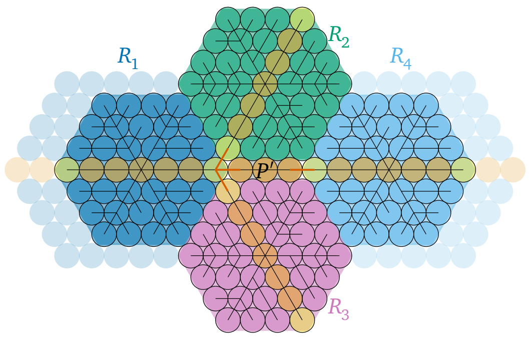


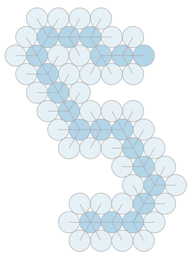
Figure 3.14: Two possible weak UDCRs of the branching gadget. The darker regions for R_1 and R_4 represent $B(3,3)$ while the combination of the darker and lighter regions for R_1 and R_4 represent $B(4,4)$. The four edges where the five parts are connected are highlighted. The two chainable hexagon trees R_2 and R_3 can be exchanged while the construction stays rigid (as shown in Lemma 3.6).

order the three children R_2, R_3, and P' of R_1 can be placed, i.e., the embedding of R_1 and its children.

Figure 3.14 shows that the counterclockwise embedding R_3, P', R_2 (and thus also R_2, P', R_3) is possible. The figure depicts $B(3,3)$ and by looking at the lighter region as well it also represents $B(4,4)$, $B(3,4)$, and $B(4,3)$. It should also be clear from the figure that for all larger radii for R_1 and R_4 the weak UDCR can be constructed in the same manner.

Due to symmetry and renaming it is now only necessary to check whether it is possible to achieve the counterclockwise embedding P', R_2, R_3 of the children of R_1. However, as we can see in Figure 3.15, placing any two of the three children does not leave enough space to place the remaining child. Here, like above, making R_1 larger does not change the way the children can be placed, it especially does not suddenly allow placing the third missing child. Since R_4 is not even in the figure, growing it does not help either. Hence, the two embeddings from Figure 3.14 are the only possible ones. $\square$

The four chainable hexagon trees R_1, R_2, R_3, and R_4 each have exactly one connector node remaining that is still a leaf node. We call them v_1, v_2, v_3, and v_4. It follows from Lemma 3.6 that the position of v_4 relative to v_1 is fixed and for the positions of v_2 and v_3 relative to v_1 and v_4 there are exactly two possibilities: Either v_2 is to the left of the line through v_1 and v_4 and v_3 is to the right of this line, or v_3 is to the left of this line and v_2 to the right of it. We can now attach line segment trees at the connector nodes to obtain more complex constructions. These constructions, however, are not necessarily rigid: Attaching another $R(3)$ to R_2 such that this $R(3)$ and R_2 form a line segment tree will result in weak UDCRs that can be partitioned into exactly



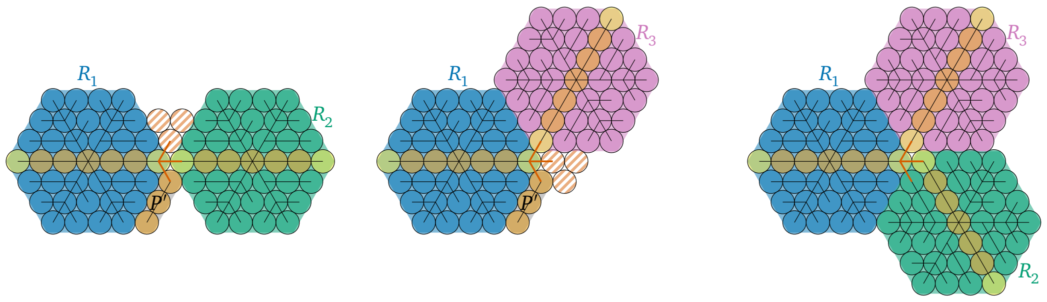


Figure 3.15: Trying to have the children of R_1 as P', R_2, R_3 in counterclockwise order does not yield a weak UDCR. After placing two of the three, there is insufficient space to place the third. In the left two cases the available space has an angle of 60° while the missing hexagon needs an angle of 120°. In the right case the grid point to connect P' is already blocked by R_2.

two equivalence classes: Those in which the additional $R(3)$ is to the left of the line through R_1, P', and R_4, and those in which it is to the right of this line. To construct the logic engine, this is exactly what we want: Exactly two possible realizations which branch off to the sides of an otherwise rigid construction.

3.2.5 Putting Everything Together

We have seen how to model line segments and four-way intersections at which the two side branches can be exchanged. One problem remains, namely, that the logic engine has a four-way intersection which branches off orthogonally while our branching gadget branches off with a 120° angle. However, this problem is easily mitigated as we can take any orthogonal logic engine, e.g., the one in Figure 3.7, and tilt all intersections and moving parts by 30°. The result can be seen in Figure 3.16: First the flags are tilted 30° upwards and then the flag poles are tilted 30° to the right. In addition, the flag poles towards the left are enlarged at the shaft such that all flags of the same level are on a line orthogonal to the flag poles.

It is easy to see that, same as before in Figure 3.7, that there are $2m$ levels and on each level there is space for $n-1$ flags. The positions of the positive and negative part of each flag pole can still be exchanged between the top and the bottom. Similarly, the flags on the leftmost and rightmost flag poles cannot be placed towards the outside of the construction and two flags of the same clause in neighboring flag poles cannot be both placed towards each other. While Figure 3.16 suggests that the flags are fully dimensional (i.e., they are two-dimensional objects), this is not necessary. We could have the flags be line segments and the construction would work exactly the same. We also added a small line segment stub to have an actual four-way intersection where the flags are attached to their flag poles.

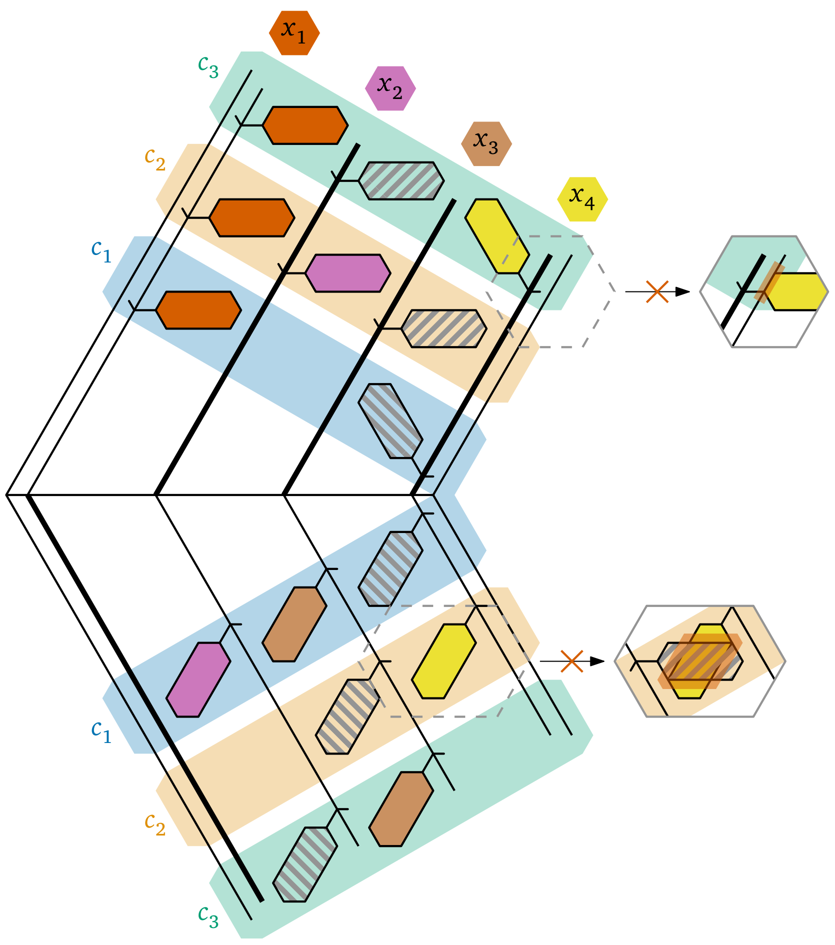


Figure 3.16: An example realization of a tilted logic engine for the NAE3SAT formula $\phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_4) \land (x_1 \lor x_3 \lor \neg x_4)$. We have taken the logic engine from Figure 3.7 and tilted all intersections and moving parts by 30°. Similar to the original logic engine the upper box on the right shows that the outermost flag poles are forced to position their flags towards the center. The lower box shows that now two flags on the same level can be positioned towards each other.

The construction now consists of line segments and four-way intersections that branch off with an angle of 120° (or 60° depending on the direction we look at). Both can be modeled by weak UDCRs of trees, as discussed in the previous sections. Given the NAE3SAT formula $\phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_4) \land (x_1 \lor x_3 \lor \neg x_4)$, one possible weak UDCR for the tree obtained from ϕ can be seen in Figure 3.17. Here, the depiction of the weak UDCR is slightly schematic in the sense that we don't see the individual unit disks or the tree edges. However, a schematic hexagon represents a weak UDCR of a tree $R(r)$ where the line segment through the hexagon represents the subtrees of type P. Where two hexagons overlap in one corner, the corresponding nodes are merged into line segment trees. Similarly to the hexagons, the line segment in the branching gadget represents the used $P(3)$. Overall, the figure shows the same realization we have previously seen in Figures 3.7 and 3.16.

We now describe in more detail on how to obtain a tree T_ϕ for a given NAE3SAT formula ϕ with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$. We will then show that T_ϕ has a weak UDCR if and only if ϕ has a valid assignment.

The shaft. The (in our figures horizontal) shaft is basically a line segment tree with $n + 1$ branching gadgets in between. We start with a branching gadget $B(3, 4)$ for the left side of the logic engine. The $R(4)$ of this gadget is reused for the next branching gadget, a $B(4, 3)$, that will be is used for x_1's flag pole. For all flag poles for x_i with $1 < i < n$ we have a $B(3, 3)$ which is joined to x_{i-1}'s branching gadget by an $R(3)$. For x_n we use $B(3, 4)$ as branching gadget, connected to x_{n-1}'s branching gadget by an $R(3)$, as well. The $R(4)$ of the branching gadget is reused for the $B(4, 3)$ used to branch off the right side of the logic engine.

Overall, the tree constructed for the shaft consists of $n+1$ branching gadgets and $n-1$ chainable hexagon trees $R(3)$ which join them together. Since both the used branching gadgets and the chainable hexagon trees $R(3)$ have constant size, the constructed tree has size $\Theta(n)$. The tree is also rigid since nothing has been added to sides of the branching gadget. Thus, the weak UDCR in Figure 3.18a, representing the resulting tree for four variables, gives us the only possible overall shape.

The flag poles. For the flag poles we construct several similar trees. Let $T_P(j)$ be the following construction for all $j \in \mathbb{N}$: We first take $2j$ copies of $R(3)$ and combine them into a line segment tree. Then we attach m overlapping branching gadgets $B(3, 3)$, i.e., R_4 of the first branching gadget is reused as R_1 of the second branching gadget, and so on. Then, for all variables x_i with $1 \leq i \leq n$ we attach $T_P(n - i)$ to both R_2 and R_3 of the branching gadget reserved for x_i on the shaft. In Figure 3.17 we can observe that adding 2 more $R(3)$ to x_i's flag pole than x_{i+1}'s flag pole results in the branching gadgets for each clause being on a line nearly orthogonal to the flag poles. This ensures that two flags of neighboring flag poles on the same level will overlap when placed towards each other.

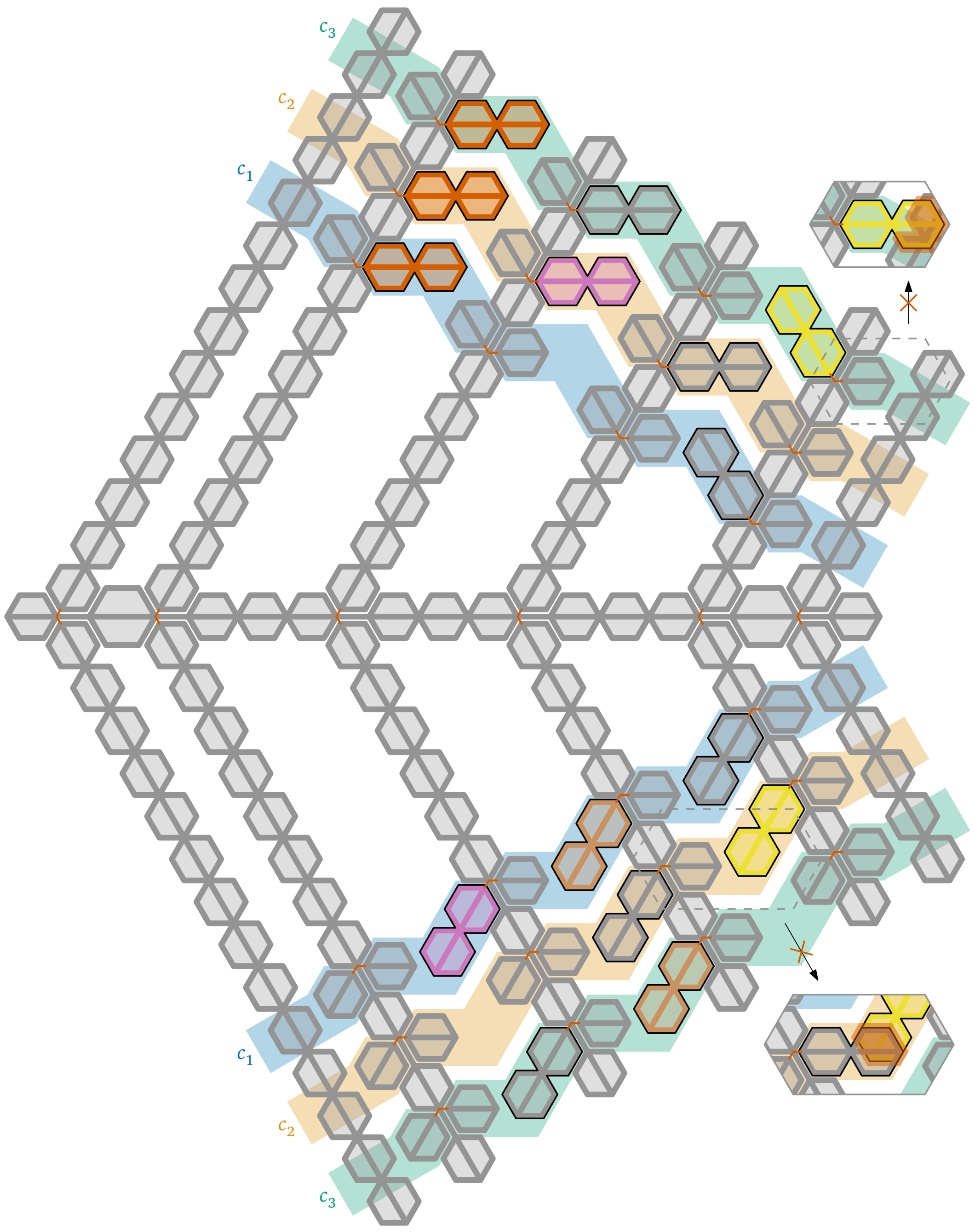
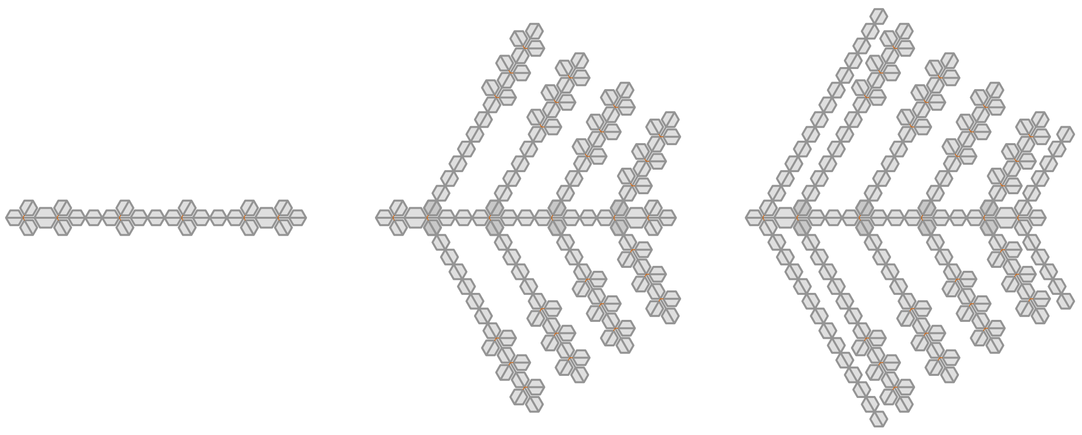


Figure 3.17: One weak UDCR of the tree constructed for the NAE3SAT formula $\phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_4) \land (x_1 \lor x_3 \lor \neg x_4)$. The black outline around the flags only highlights them and has no additional meaning. The hexagons are all $R(3)$ except for the two larger $R(4)$. The small orange 120° angles indicate where the branching gadgets connect R_2 and R_3 to R_1. The additional boxes show that unwanted flag placement cannot happen due to resulting overlaps.



(a) The shaft is constructed in the first step.

(b) In the next step we add the flag poles.

(c) In the third step we add the sides.

Figure 3.18: Weak UDCRs of the step-wise construction of the logic engine from Figure 3.17 with four variables and three clauses. The constructed tree is rigid during these three steps.

In total, the flag poles need $2mn$ branching gadgets as well as $2n^2 - 2n$ copies of $R(3)$. Thus, we add $\Theta(n^2 + mn)$ additional nodes to the tree. As the flag poles themselves are rigid and we always add two identical ones to the same branching gadget on the shaft, the construction stays rigid. In Figure 3.18b we see a weak UDCR for a resulting tree with four variables and three clauses; all possible weak UDCRs have the shape shown there.

The sides. After adding the flag poles we still need to add the left and right side of the logic engine. The criterion here is that they need to be as long as x_1's and x_n's flag pole, respectively. By counting the length of overlapping branching gadgets we can see that m overlapping branching gadgets have length $m(7+3)+7 = 10m+7$, since it is constructed by combining $m+1$ copies of $R(3)$ and m copies of $P(3)$. On the other hand, taking k copies of $R(3)$ as a line segment tree will result in length $6k+1$. We can observe that for every m divisible by 3 we can construct a line segment tree out of $k = \frac{5}{3}m + 1$ copies of $R(3)$ which results in length $10m+7$, the same as m overlapping branching gadgets. We can thus conclude that constructing a line segment tree out of $\left\lceil \frac{5}{3}m \right\rceil + 1$ copies of $R(3)$ will be at least as long as the shortest flag pole.

The right side of the logic engine can be shorter by one $R(3)$ due to the tilting. We can thus add two line segment trees consisting of $\left\lceil \frac{5}{3}m \right\rceil$ copies of $R(3)$ to the rightmost branching gadget to represent the logic engine's right side. For the left side, the construction uses

$$\left\lceil \frac{5}{3}m \right\rceil + 2(n-1) + 1 = \left\lceil \frac{5}{3}m \right\rceil + 2n - 1$$

copies of $R(3)$: $\left\lceil \frac{5}{3}m \right\rceil$ for the height of the overlapping branching gadgets, $2(n-1)$ for the additional height of x_1's flag pole and one additional $R(3)$ to account for the tilting.

Overall we need $\Theta(m+n)$ nodes to construct the trees representing the logic engine's sides. In Figure 3.18c we see the weak UDCR resulting from the constructed tree so far for four variables and three clauses. As we add two identical line segment trees to both sides of each branching gadget, the resulting tree stays rigid and thus the shape shown in the figure is the only possible one.

The flags. The flags in Figure 3.17 seem to consist of two $R(3)$, however one of them is part of the branching gadget and thus, already counted. As a result, for every flag we add to the construction, we need one additional $R(3)$. As described in Section 3.2.1, for every combination of variable x_i and clause c_j we add two flags if x_i does not appear in c_j and one flag otherwise. Thus, a total of fewer than $2mn$ flags are added; the exact amount is between $2mn - 3m$ and $2mn - 2m$.[1] Thus, another $\Theta(mn)$ nodes are added to the tree by the flags.

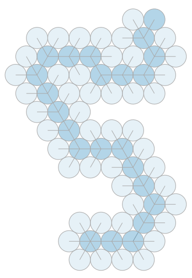
Since we only add an $R(3)$ to one side of a branching gadget for every flag, the whole construction becomes non-rigid. However, since the branching gadgets allow only two possible non-congruent weak UDCRs, there are only two ways a flag can be placed: to the left or the right of the branching gadget.

Having described the full construction we can now finalize this section with

Theorem 3.2. *The problem of finding a weak UDCR for a tree is NP-hard.*

Proof. As already described at the beginning of Section 3.2, we will reduce to our problem from Not-All-Equal 3SAT (NAE3SAT). Given a Boolean 3SAT formula ϕ with n variables and m clauses we know that there is a valid assignment of the variables (and thus ϕ is a yes instance for NAE3SAT) if and only if the corresponding logic engine construction (as presented by Eades and Whitesides [EW96]) can be realized in the plane without overlaps. We have argued in Section 3.2.1 that a simpler logic engine is functionally equivalent and in the beginning of Section 3.2.5 argued that we can even tilt the moving parts without any difference in functionality. This tilted version of the logic engine can then be modeled by a weak UDCR of a tree T_ϕ that is constructed from the given formula ϕ. We have also shown that T_ϕ has a weak UDCR if and only if the logic engine for ϕ can be realized in the plane without overlaps. Thus, we can conclude that ϕ is a yes instance for Not-All-Equal 3SAT (NAE3SAT) if and only if T_ϕ has a weak UDCR. $\hookrightarrow$

[1] Every clause needs at least 2 distinct variables (otherwise there cannot be a true and a false literal in it), placing at least $2m$ times just one flag; on the other hand it has at most 3 distinct variables, placing at most $3m$ times just one flag.



It remains to show that what we described is, indeed, a polynomial time reduction. We described during the construction of T_ϕ that we use $\Theta(n)$ nodes for the shaft, $\Theta(n^2 + mn)$ nodes for the flag poles, $\Theta(m + n)$ nodes for the sides, and $\Theta(mn)$ nodes for the flags. The total number of nodes, and thus the size of T_ϕ is in $\Theta(n^2 + mn)$. To construct T_ϕ we do not need to perform any additional work, hence the running time of the reduction is in $\Theta(n^2 + mn)$ as well. □

After showing that the problem is NP-hard, the obvious question is whether the problem is also contained in NP and thus NP-complete. However, we do not have an answer to this question. What we can say is that the problem is contained in $\exists\mathbb{R}$. This is the class of all problems which can be reduced in polynomial time to a Boolean formula of the form $\exists x_1, \ldots, x_n \in \mathbb{R} : F(x_1, \ldots, x_n)$. Here, F is a quantifier-free formula that allows logically combining equalities and inequalities of polynomials of the variables x_1 to x_n. It is easy to see that our problem is contained in $\exists\mathbb{R}$ with the following reduction.

Given a graph, introduce one pair of variables x_v and y_v for each node v. For each pair of nodes u, v such that u and v are connected by an edge we add the constraint $(x_u - x_v)^2 + (y_u - y_v)^2 = 2^2$ to ensure that their disks touch. For all other pairs u, v we add $(x_u - x_v)^2 + (y_u - y_v)^2 \geq 2^2$ to ensure that no two disks intersect. Thus, given a graph with n nodes we construct a formula with $2n$ variables and $\Theta\left(n^2\right)$ (in)equalities of constant size, resulting in a polynomial size formula. It is also easy to see that the constraints model exactly the requirements from Definition 2.2. The set of disks is interior-disjoint (for all pairs of disks their centers have distance at least 2) and adjacent disks must touch (their center have distance exactly 2).

If, however, we are only interested in finding a grid-restricted weak UDCR, we can show that the problem is contained in NP. Given a graph the certificate of a grid-restricted weak UDCR gives us the disk center for each node. Since the disks centers are all on the hexagonal grid, their coordinates can be given as integers. The verification algorithm then only needs to check that no two disks are placed on the same grid position and that the disks of adjacent nodes are placed on adjacent grid positions. This can be easily done in quadratic time by checking all pairs of nodes. This approach does not work if the weak UDCRs are not grid-restricted, since we cannot be sure that the disk coordinates have polynomial size in the input.

We can now conclude this chapter with a summary of our results. In Section 3.1 we showed a linear-time algorithm for finding weak UDCRs for caterpillars. The same problem is NP-hard for trees, as shown in Section 3.2, by giving a reduction from NAE3SAT. We showed that the problem is in $\exists\mathbb{R}$ and in the grid-restricted case it is even NP-complete.

Chapter 4

Graphs With a Fixed Embedding

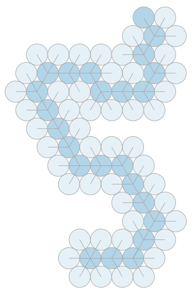
In the previous chapter we have shown results for weak UDCRs where for all disks the order of its neighbors could be chosen arbitrarily. Now, we will look at the situation in which we are not only given a graph G but also a combinatorial embedding which needs to be respected by the weak unit disk contact representation. We defined such weak embedded unit disk contact representations (weak embedded UDCRs) already in Definition 2.3. Similar to Chapter 3, this chapter also contains two main results: In Section 4.1 we present a linear time algorithm to find weak embedded UDCRs for grid-representable, strictly x-monotone caterpillars while in Section 4.2 we show that the problem becomes NP-hard for general caterpillars. The algorithm for weak UDCRs of caterpillars presented in Section 3.1 cannot be used for graphs with a fixed embedding since it relies heavily on choosing which side of the caterpillar's spine the leaf disks are placed on. If we cannot place the leaf disks freely, the spine disks can be forced to not lie on a line. This will be used in the NP-hardness proof in Section 4.2. These results were previously published in short form in [CCN19].

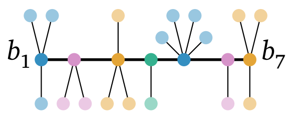
4.1 A Linear Time Algorithm for Grid-Representable, Strictly x-Monotone Caterpillars

Finding a weak embedded UDCR for a path is trivially done in linear time by placing all disks on a line. This is especially easy, as there can be no two different embeddings for paths. Additionally, even for some caterpillars and embeddings we can find a weak embedded UDCR in linear time, namely for grid-representable, strictly x-monotone ones, where the latter is defined as follows.

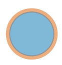
Definition 4.1. Let G be a caterpillar with backbone nodes $b_1, \ldots, b_k$ in order, and let Γ an embedding for G. Let $\mathcal{D}$ be a weak embedded UDCR for G, such that for all $1 \leq i \leq k$, the disk $D_i \in \mathcal{D}$ corresponds to backbone node b_i. Then $\mathcal{D}$ is *strictly x-monotone* if for all $1 < i \leq k$ it holds true that D_i's center is strictly to the right of D_{i-1}'s center. The caterpillar G is *strictly x-monotone* if and only if it has a strictly x-monotone weak embedded UDCR.

▷ strictly x-monotone

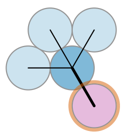




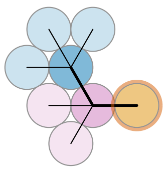
(a) G has 7 backbone nodes.



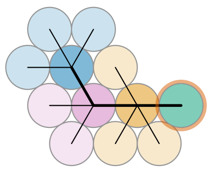
(b) $\mathscr{D}_0$.



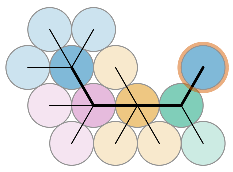
(c) $\mathscr{D}_1$.



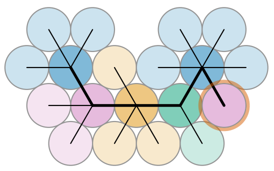
(d) $\mathscr{D}_2$.



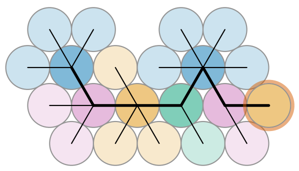
(e) $\mathscr{D}_3$.



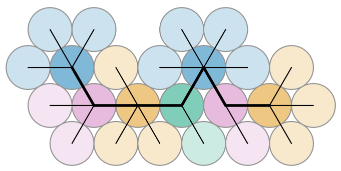
(f) $\mathscr{D}_4$.



(g) $\mathscr{D}_5$.



(h) $\mathscr{D}_6$.



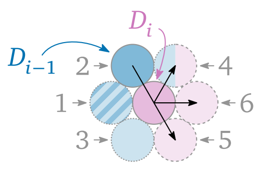
(i) $\mathscr{D}_7 = \mathscr{D}$.

Figure 4.1: (a) An example caterpillar with embedding and (i) a possible weak embedded UDCR $\mathscr{D} = \mathscr{D}_7$. In (b–h) we see the intermediate weak embedded UDCRs $\mathscr{D}_i$ with backbone disk D_{i+1} highlighted, for all $0 \leq i \leq 6$. Backbone disks have a darker and leaf disks have a lighter color.

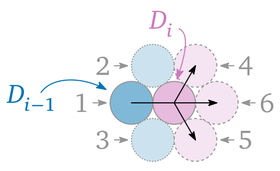
Similarly to the algorithm in Section 3.1.2 for the non-embedded caterpillars, the algorithm will find grid-restricted weak embedded UDCRs. The idea of the algorithm is simple: To find a weak embedded UDCR $\mathscr{D}$ for a given caterpillar G with k backbone nodes, we start with a 0-restricted weak embedded UDCR $\mathscr{D}_0$ of only the first backbone disk.[1] We then extend it to a 1-restricted weak embedded UDCR $\mathscr{D}_1$ for G_1, a 2-restricted weak embedded UDCR $\mathscr{D}_2$ for G_2, and so on, until we have obtained a weak embedded UDCR $\mathscr{D}_k = \mathscr{D}$ for $G_k = G$. See Figure 4.1 for an example caterpillar together with one weak embedded UDCR and all intermediate steps.

However, instead of extending an i-restricted weak embedded UDCR to exactly one $(i+1)$-restricted weak embedded UDCR the algorithm needs to try out all possible ways to extend it. For all those extensions it needs to try out all possible extensions and so on. Naively, this would result in an exponential amount of work, but we will see that it can be expressed as a dynamic program that runs in linear time.

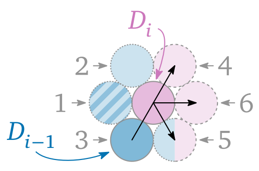
[1]Remember that $\mathscr{D}_i$ is the i-restricted weak embedded UDCR of the i-restricted caterpillar G_i as defined in Definitions 3.1 and 3.2.



(a) The bottom left disk may only be occupied by a leaf disk of the hatched disk at 1_i (which would be D_{i-2}).



(b) In this situation only up to three positions next to D_i can be occupied by $\mathscr{D}_{i-1}$.



(c) The same situation as (a) but vertically mirrored.

Figure 4.2: Possible situations for grid-restricted, strictly x-monotone weak embedded UDCRs: the dark blue disk D_{i-1} is to the left of dark purple disk D_i. The light blue disks show possible locations of disks of $\mathscr{D}_{i-1}$ while the light purple disks show possible locations for D_{i+1} (if it exists).

Before describing the dynamic program, we will first make some observations about grid-restricted, strictly x-monotone weak embedded UDCRs that will help us in understanding the mechanics the dynamic program will exploit.

4.1.1 Preliminary Observations

Since we talk about grid-restricted weak embedded UDCRs, it is obvious that for any disk there are exactly six possible grid locations where a touching disk can be placed. These six locations are identified by the numbers 1 to 6 (from left to right and top to bottom) as already seen in Figure 3.2a. We will often talk about such a position relative to a disk. To do this we use the notation 1_i to refer to position 1 relative to disk D_i. The same is true for all other positions 2 to 6. With this, we are now able to show some properties of grid-restricted, strictly x-monotone weak embedded UDCRs:

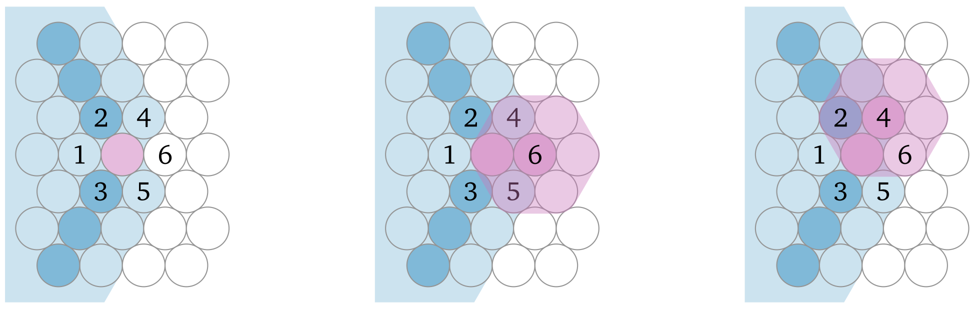
Lemma 4.1. *Let $\mathscr{D}$ be a grid-restricted, strictly x-monotone weak embedded UDCR of a caterpillar G with backbone nodes $b_1, \dots, b_k$ in this order. Let $D_i \in \mathscr{D}$ be the disk for b_i for all $1 \leq i \leq k$. In addition, for all $0 \leq i \leq k$, let G_i be the i-restricted caterpillar of G and $\mathscr{D}_i$ be the i-restricted weak embedded UDCR. Then for all $1 < i \leq k$ disk D_{i-1} is in position 1_i, 2_i, or 3_i.*

In addition, depending on D_{i-1}'s position, other positions may be occupied by disks in $\mathscr{D}_{i-1}$: for D_{i-1} at 1_i those are 2_i and 3_i; for D_{i-1} at 2_i they are 1_i, 3_i, and 4_i; and for D_{i-1} at 3_i they are 1_i, 2_i, and 5_i.

If D_{i-1} is in position 2_i and 3_i is occupied, or D_{i-1} is in position 3_i and 2_i is occupied, then 1_is also occupied.

Proof. The first statement follows directly from Definition 4.1 since D_i must be strictly to the right of D_{i-1} and thus D_{i-1} strictly to the left of D_i.

↪



(a) The blue positions are the only positions that can be occupied when the purple backbone disk is placed.

(b) Placing the next backbone disk at position 6: blocked disks can be determined directly.

(c) Placing the next backbone disk at position 4: there is one unlabeled blue disk that may be blocked.

Figure 4.3: Blue positions may contain a disk, while white position are always empty. (a) Around the current backbone disk, all positions but 6 may be blocked. (b) If the next backbone disk is placed at position 6 it is easy to determine the blocked positions. (c) Placed at position 4 there is one unlabeled blue position.

For the second statement we look at Figure 4.2. For position 1_i (Figure 4.2b) we see that the only direct neighbors of D_{i-1} can be positions 2_i and 3_i. In addition, since all previous backbone disks lie strictly to the left of D_{i-1} they cannot be direct neighbors of the other positions, either. For position 2_i (Figure 4.2a) we see that the direct neighbors are 1_i and 4_i. However, 1_i may be occupied by backbone disk D_{i-2} and thus its leaf disk may occupy 3_i. Only a leaf disk can occupy 3_i as all backbone disks must be to the left of D_{i-1} at 2_i; thus 5_i cannot be occupied. Additionally, the only positions which can be occupied by backbone disks such that one of their leaf disks is placed at 3_i are 1_i or the position to the left of 3_i. In the latter case, 1_i must also be occupied by a backbone disk as it lies on the only strictly x-monotone grid path from 2_i to the position left of 3_i. Thus, if 3_i is occupied, 1_i is also occupied (by a backbone disk), proving the third statement.

The second and third statement also hold for D_{i-1} at position 3_i by symmetry. □

Corollary 4.1. *From the previous lemma it follows that the set of occupied disks around D_i in $\mathscr{D}_{i-1}$ is contiguous. More specifically, given the sequence $S = (4_i, 2_i, 1_i, 3_i, 5_i)$ of positions around D_i, excluding 6_i, those positions in S occupied by disks in $\mathscr{D}_{i-1}$ form a contiguous subsequence of S. Since 6_i is always free, this also implies that the set of free positions around D_i is contiguous: More specifically, the subsequence of free positions in $S' = (3_i, 5_i, 6_i, 4_i, 2_i)$ is contiguous.* □

In order to know where we can place neighbors of D_i, is it sufficient to know which of the six positions around it are occupied. In addition, since all previous backbone disks lie to the left of D_i, and the next backbone disks are placed to the right of D_i they should not interfere (too much). See Figure 4.3a where we show in dark blue the most extreme previous backbone disk positions and in light blue all positions that may be occupied (never all at the same time) when placing the central purple disk D_i. The white positions are always free. When placing the next backbone disk D_{i+1} at position 6_i, as seen in Figure 4.3b, we can compute which of the six positions are occupied around D_{i+1} from the knowledge which of the six positions around D_i are occupied. This can be seen by the fact that the non-white positions inside the shaded hexagon are all labeled with the positions around D_i or are D_i itself. If, however, we place D_{i+1} at position 4_i, there is one blue position (above 2_i and 4_i) inside the shaded hexagon. That is, we cannot compute all occupied positions around D_{i+1} only from the positions around D_i. We can include this missing position (and the symmetric one below 3_i and 5_i) in the positions that we keep track of, which results in a rhombus shaped region which we can see in Figure 4.4a. If we extend the shaded hexagons in Figures 4.3b and 4.3c by one disk to the top and bottom we also see that only white or already labeled disk are added. We can thus conclude

Observation 4.1. *While incrementally constructing a grid-restricted, strictly x-monotone weak embedded UDCR, it is sufficient to know which of the eight positions depicted in Figure 4.4a around a backbone disk are occupied, to calculate the same information for the next backbone disk.* □

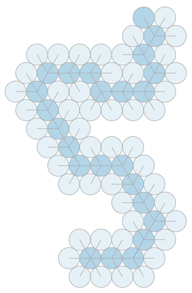
4.1.2 The Linear Time Algorithm

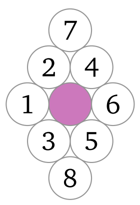
As described before, the idea of the algorithm is to try out all possible ways to incrementally construct a grid-restricted, strictly x-monotone weak embedded UDCR for a given caterpillar G with its embedding Γ and k backbone nodes $b_1, \ldots, b_k$. For this we need to know whether an existing i-restricted weak embedded UDCR can be extended to a full weak embedded UDCR. According to Observation 4.1 the only information we need for this is which of the eight positions around D_i are occupied. We thus formulate the algorithm as a dynamic program

$$\text{Realizable} : [1..k] \times 2^{[1..8]} \to \{\text{true}, \text{false}\}.$$

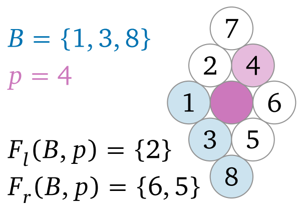
The first input i, with $1 \leq i \leq k$, tells us which backbone node b_i we are currently considering. The second input, a subset of [1..8], tells us, which of the eight positions around the already placed disk D_i are blocked by disks from $\mathcal{D}_{i-1}$ corresponding to G_{i-1}. Given this information we should then return true if and only if $\mathcal{D}_{i-1}$ can be extended to a weak embedded UDCR $\mathcal{D}$ of G.

However, to formulate and understand the dynamic program we need some additional definitions:

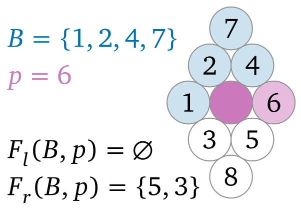




(a) Eight positions around the center disk are considered by the dynamic program.



(b) Three blocked positions and position 4 for the next backbone node.



(c) Four blocked positions and position 6 for the next backbone node.

Figure 4.4: The dynamic program needs to consider the eight positions shown in (a). In (b) and (c) we see example situations from the algorithm with different blocked positions and different position for the next backbone disk.

Definition 4.2. Let G be a caterpillar with backbone nodes $b_1, \dots, b_k$ in this order and an embedding Γ. Then, let n_i be the number of leaf neighbors of b_i for all $1 \leq i \leq k$. For all $1 < i < k$ let l_i be the number of leaf neighbors of b_i between b_{i+1} and b_{i-1} in counterclockwise order according to Γ. These are the leaves to the left of a line from b_i through b_{i+1} in any drawing of G respecting Γ. Similarly, let r_i be the same but in clockwise order, which are those leaves to the right. For $i \in \{1, k\}$ let $l_i = n_i$ and $r_i = 0$.

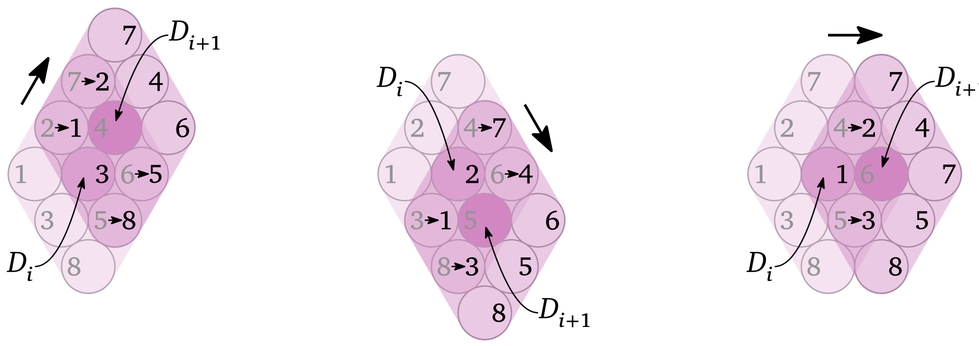
Given $B \subseteq [1..8]$ and $p \in \{4,5,6\}$ let $F(B,p) = [1..6] \setminus \big(B \cup \{p\}\big)$ the set of all free positions around the center. Then, let $F_l(B,p)$ be the contiguous set of all free positions counterclockwise of p, i.e., we start one position counterclockwise of p, continue counterclockwise until we hit the first blocked position, and collect all encountered free positions as $F_l(B,p)$. $F_r(B,p)$ is the same but in clockwise direction.

See Figures 4.4b and 4.4c for two examples of the latter definitions. In Figure 4.4b we have $B = \{1,3,8\}$ and $p = 4$, and thus, $F(B,p) = \{2,6,5\}$. To compute $F_l(B,p)$ we start at position 2 and then already encounter the blocked position 1. For $F_r(B,p)$ we start at 6, continue to 5 and then encounter blocked position 3. Similarly, in Figure 4.4c there are no free positions counterclockwise of position 6 and two free positions clockwise.

As previously discussed, Observation 4.1 tells us that the knowledge, which of the eight positions from Figure 4.4a around a backbone disk are occupied, is sufficient to know which positions around the next backbone disk are occupied. However, to describe our algorithm we must be able to explicitly compute them. To do this, we introduce the function

$$\text{shift} : 2^{[1..8]} \times \{4,5,6\} \to 2^{[1..8]}.$$

Given a set of occupied positions and the position of the next backbone disk, it returns the set of occupied positions around the next backbone disk. The function is easily



(a) The new backbone disk is placed at position 4 relative to the old backbone disk, see also Eq. (4.1).

(b) The new backbone disk is placed at position 5 relative to the old backbone disk, see also Eq. (4.2).

(c) The new backbone disk is placed at position 6 relative to the old backbone disk, see also Eq. (4.3).

Figure 4.5: The blocked positions around the old backbone disk D_i are taken to obtain the blocked positions around the new backbone disk D_{i+1}. This effectively shifts the rhombus of blocked positions by the arrow. The gray labels on the left indicate positions relative to D_i while the dark labels on the right indicate positions relative to D_{i+1}.

described as

$$\mathrm{shift}(B,4) = \{3\} \cup \{1 \mid 2 \in B\} \cup \{2 \mid 7 \in B\} \cup \{5 \mid 6 \in B\} \cup \{8 \mid 5 \in B\} \tag{4.1}$$

$$\mathrm{shift}(B,5) = \{2\} \cup \{1 \mid 3 \in B\} \cup \{3 \mid 8 \in B\} \cup \{4 \mid 6 \in B\} \cup \{7 \mid 4 \in B\} \tag{4.2}$$

$$\mathrm{shift}(B,6) = \{1\} \cup \{2 \mid 4 \in B\} \cup \{3 \mid 5 \in B\} \tag{4.3}$$

where $\{a \mid b \in B\}$ is $\{a\}$ if $b \in B$ and otherwise the empty set. A depiction to understand better which previous positions imply which new positions can be found in Figure 4.5.

We now describe our dynamic program Realizable(i, B), whose pseudocode can be found in Algorithm 4.1. To see whether a given caterpillar has a weak embedded UDCR we call Realizable$(1, \varnothing)$ and check whether the result is true or false.

At first, we look at the base case for $i = k$ in lines 2 to 3. Here, we only check whether the occupied positions directly around D_k together with the n_k remaining leaf disks to be placed are not more than six. The recursive case (lines 4 to 10) is more involved. First, we will try to place D_{i+1}, the next backbone disk, at all three possible positions that are not occupied (line 5). For this placement of D_{i+1} at position p, in line 6 we then obtain all free positions counterclockwise of p and iterate through all possibilities to place the l_i left leaf nodes of b_i among those free positions.[2] The same is done in line 7 for the positions and leaf nodes to the right. Then, in line 8, the algorithm calculates which positions are occupied relative to the next backbone disk from the

[2]We use the notation $[A]^b$ to mean $[A]^b = \{C \subseteq A \mid |C| = b\}$.

Algorithm 4.1: The dynamic program to find out whether a caterpillar G with k backbone nodes has a grid-restricted, strictly x-monotone weak embedded UDCR.

```
1:  function Realizable($i, B$)
2:      if $i = k$ then                       ▷ The base case; no more backbone disks to place.
3:          return $n_k \leq 6 - |B \cap [1..6]|$   ▷ Check for sufficient free positions for leaves.
4:      else                                  ▷ The recursive case with $i < k$.
5:          for all $p \in \{4,5,6\} \setminus B$ do     ▷ All free and allowed positions for $D_{i+1}$.
6:              for all $P_l \in [F_l(B,p)]^{l_i}$ do    ▷ Possible placements to the left.[2]
7:                  for all $P_r \in [F_r(B,p)]^{r_i}$ do    ▷ Possible placements to the right.[2]
8:                      $B' \leftarrow \text{shift}(B \uplus P_l \uplus P_r \uplus \{p\}, p)$   ▷ Obtain new blocked positions.
9:                      if Realizable($i+1, B'$) then   ▷ If a valid extension is found …
10:                         return true              ▷ … we can return successfully.
11:     return false
```

occupied positions B and the neighboring disks placed at positions P_l, P_r, and p. If the current placement can be extended to a weak embedded UDCR, which is checked by recursively calling R for the next backbone node with the new blocked positions in line 9, then we can return successfully in line 10. If, however, none of the placements result in a weak embedded UDCR, we finally return that no weak embedded UDCR exists in line 11.

4.1.3 Running Time and Correctness

Since R is a dynamic program, it is evaluated at most once for every possible input. The number of possible inputs is at most $2^8 = 256$ for the set B and at most k for the current backbone node i. Thus, at most $O(k)$ different inputs exist. To bound the running time of each call to R we observe that the base case obviously runs in constant time. For the recursive case, the loop in line 5 runs at most three times. Both loops in lines 6 and 7 run as often as there are possibilities to place leaf disks of the current backbone disk among the free positions. However, $0 \leq l_i, r_i \leq 4$ and the number of free positions to the left and right is at most 4 as well. That is, the number of times each loop runs is never larger than $\max_{0 \leq a,b \leq 4} \binom{a}{b} = 6$. Thus, the number of times lines 8 and 9 are executed is constant, and both lines can be computed in constant time, assuming that Realizable$(i+1, B')$ has already been calculated.

We can thus conclude that the dynamic program runs in $O(k)$ time if the caterpillar with its embedding is given as a list of k pairs with the numbers of left and right leaf nodes. If, however, the caterpillar is given as a graph with ordered neighbors, we first need to compute these numbers which takes an additional $O(n)$ time. Thus, the running time of the dynamic program is always linear in the size of the given input or more generally $O(n)$ as $k \leq n$.

To show correctness we want to see that the algorithm returns “true” if and only if the presented caterpillar with its embedding has a grid-restricted, strictly x-monotone weak embedded UDCR:

Lemma 4.2. *Let G be a caterpillar with embedding Γ and backbone nodes $b_1, \dots, b_k$ in this order. Then the R finds a grid-restricted, x-monotone weak embedded UDCR if and only if G has a grid-restricted, x-monotone weak embedded UDCR.*

Proof. We first show that R only returns “true” if it has actually found a grid-restricted, x-monotone weak embedded UDCR. Since we only ever place disks at one of the six grid locations around the backbone disks, the resulting construction is grid-restricted. In addition, the next backbone disks are always placed strictly to the right of the current backbone disk, making the construction strictly x-monotone. From Observation 4.1 we know that the set B passed to R is sufficient to calculate the same information for the next backbone disk and thus, using induction, for all following backbone disks. This means, that we never assume a position to be free while it is actually occupied. In addition, as suggested in line 8 of Algorithm 4.1 the selected positions for the next backbone disk and the left and right leaf disks are all disjoint from each other and the already occupied positions. This means that no two disks are ever placed at the same position. Thus, if R returns “true” it has actually found a grid-restricted, x-monotone weak embedded UDCR.

We now show that the dynamic program also finds a grid-restricted, x-monotone weak embedded UDCR if one exists. Let $\mathcal{D}$ be such a weak embedded UDCR. We show by induction over i that for any i-restricted weak embedded UDCR $\mathcal{D}_i$ of $\mathcal{D}$ the dynamic program will try placing the disks exactly as in $\mathcal{D}_i$. For $i = 1$ we know that $B = \emptyset$ in R. In line 5, the dynamic program then tries our all three possible positions of D_2 one of which must be its position in $\mathcal{D}_1$. Since $B = \emptyset$ we have that $F_l(B, p) = F_r(B, p) = [1..6] \setminus \{p\}$ and from Definition 4.2 we know that $l_1 = n_1$, the number of leaf nodes of b_1 and $r_1 = 0$. Thus, line 6 enumerates all possible ways to place the n_1 leaf disks while line 7 loops exactly once with $P_r = \emptyset$ since $[F_r(B, p)]^{r_1} = [F_r(B, p)]^0 = \{\emptyset\}$. Hence, Realizable$(1, \emptyset)$ tries out all possible placements for the second backbone disk together with all possible placements of the leaf disks. $\mathcal{D}_1$ must be among all those possibilities.

For the inductive step, i.e., $1 < i < k$, we can assume that $\mathcal{D}_{i-1}$ has one corresponding selection of p, P_l, P_r in Realizable$(i-1, B_{i-1})$ for B_{i-1} depending on $\mathcal{D}_{i-2}$ (or $B_{i-1} = \emptyset$ if $i = 2$). This corresponding selection results in calling Realizable(i, B_i) with $B_i = \mathrm{shift}(\mathrm{B}_{\mathrm{i\text{-}1}} \uplus \mathrm{P}_\mathrm{l} \uplus \mathrm{P}_\mathrm{r} \uplus \{\mathrm{p}\}, \mathrm{p})$. It also means that the positions blocked around D_i in $\mathcal{D}_{i-1}$ are exactly the positions in B_i. We need to show that Realizable(i, B_i) will try out the placement of the additional neighboring disks of D_i that result in $\mathcal{D}_i$. Again, the dynamic program tries out all three positions for D_{i+1} that are available, among them the one in $\mathcal{D}_i$. It then tries out all possible placements of left leaf disks in the free positions to the left of D_{i+1}. Here there could be a problem, since we only look at the

↪

free positions until the first occupied position. If there were any free positions between the occupied positions, we may not try the placement in $\mathcal{D}_i$. However, Corollary 4.1 tells us that the set of occupied disks is contiguous and thus line 6 tries out all possible free positions counterclockwise between D_{i+1} and D_{i-1}. The same arguments apply for the leaf disks and free positions on the right; line 7.

Finally, for $i = k$ the dynamic program just checks whether there are sufficient free positions to place all remaining leaf disks. This implicitly allows for all possible placements and thus especially for the one in $\mathcal{D}_k = \mathcal{D}$. □

Combining the running time analysis at the beginning of this section with Lemma 4.2 we can conclude with

Theorem 4.1. *Given a caterpillar G with n nodes, the dynamic program in Section 4.1.2 runs in $\Theta(n)$ time and finds a grid-restricted, strictly x-monotone weak embedded UDCR for G if and only if G has such a weak embedded UDCR.* □

4.2 NP-hardness for General Caterpillars

The algorithm presented in the previous section only works for restricted caterpillars, that is, those with a grid-restricted, strictly x-monotone weak embedded UDCR. The reason is that, for general caterpillars, the problem becomes NP-hard. We will show hardness via a reduction from Planar 3SAT.

4.2.1 Planar 3SAT

As the name suggests, Planar 3SAT is a 3SAT variant. The planarity is a property of the graph associated with a given 3SAT formula, which is defined as follows:

▷ associated graph

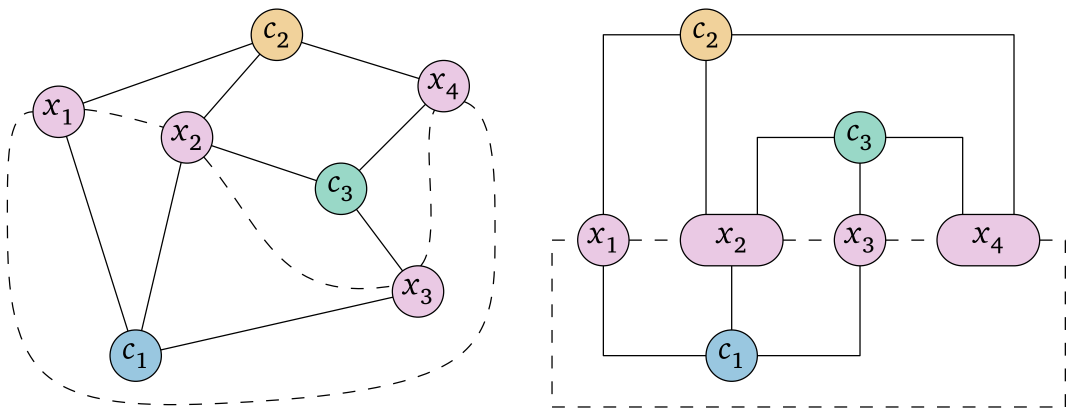
Definition 4.3. Let ϕ be a 3SAT formula with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$. We call the graph $G(\phi) = (V, E)$ the *associated graph* of ϕ where

- $V = \{x_1, \dots, x_n\} \cup \{c_1, \dots, c_m\}$,
- for all $1 \le i \le n$ and $1 \le j \le m$ we have that $\{x_i, c_j\} \in E$ if x_i appears in c_j as either x_i or $\neg x_i$,
- $\{x_i, x_{i+1}\} \in E$ for all $1 \le i < n$, and additionally $\{x_n, x_1\} \in E$.

If the associated graph of a formula ϕ is planar then this is an instance of the Planar 3SAT problem:

▷ Planar 3SAT

Definition 4.4. Given a 3SAT formula ϕ and its associated graph $G(\phi)$ together with a planar embedding, *Planar 3SAT* is the problem of deciding whether ϕ is satisfiable.



(a) A planar embedding of $G(\phi)$. The cycle through the variables is dashed.

(b) A rectilinear drawing of the planar embedding in Figure 4.6a.

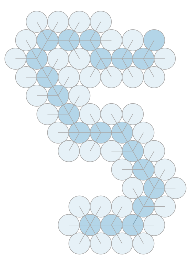
Figure 4.6: Given $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$ we see its associated graph $G(\phi)$ and two different representations.

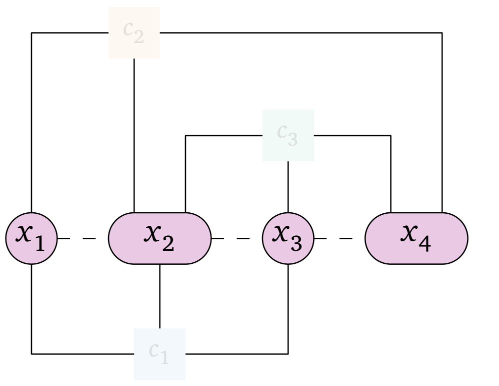
An example of a planar drawing of an associated graph can be found in Figure 4.6a. It was shown by Lichtenstein [Lic82] that Planar 3SAT remains an NP-hard problem. See Tippenhauer [Tip16] for an overview of various variations of Planar 3SAT. Knuth and Raghunathan [KR92] use a slightly different construction from which we can deduce that it is always possible to rearrange the planar drawing of an associated graph such that all variables are on a horizontal line and all edges are rectilinear. A rectilinear drawing of the associated graph from Figure 4.6a is shown in Figure 4.6b. Given this rectilinear drawing, we construct a path that follows nearly all edges present: First, remove the clause nodes and the edge $\{x_n, x_1\}$ from the drawing and leave everything else, especially the edges connecting the variable to the (now removed) clause nodes in place. The result for our example can be seen in Figure 4.7a. We can now start at the leftmost variable node and trace around all edges and stop once we could close the path to a cycle. See Figure 4.7b for the resulting path; here we also highlight the positions where the variable and clause nodes were placed previously.

Constructing this path will be the first step of reducing from Planar 3SAT to our problem of finding a weak embedded UDCR for a given caterpillar. The idea is that we will be able to construct a caterpillar which, when realized as a weak embedded UDCR, is forced to follow the path in Figure 4.7b. With some additional gadgets, which we will develop in the next sections, we will be able to find a weak embedded UDCR for the caterpillar constructed from a 3SAT formula ϕ if and only if ϕ is satisfiable.

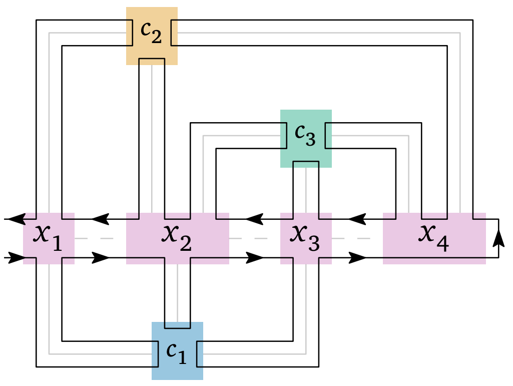
4.2.2 Rigid Caterpillars and Caterpillars With Two Representations

To be able to construct a caterpillar whose weak embedded UDCR follows the path as described in the previous definition and depicted in Figure 4.7b the caterpillar needs





(a) In the first step we remove the edge $\{x_4, x_1\}$ and the three clause nodes from Figure 4.6b.



(b) The path is constructed by tracing around all edges. The parts at which the nodes were placed are highlighted accordingly.

Figure 4.7: How to construct the path from the rectilinear drawing Figure 4.6b.

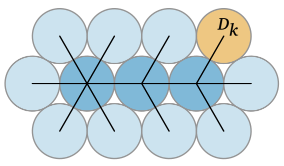
to be rigid as previously defined in Definition 2.4. The ideas are similar to the ones we have already seen for graphs without embedding in Section 3.2.2. It is easy to see that a certain type of caterpillar is rigid:

Lemma 4.3. *Let G be a caterpillar with backbone nodes $b_1, \ldots, b_k$, $k \geq 1$, and let Γ be a combinatorial embedding of G. If G has a weak embedded UDCR with respect to Γ then G is rigid and all weak embedded UDCR can be placed on the triangular grid if* $\deg(b_1) = 6$ *and* $\deg(b_i) = 4$ *for all $1 < i \leq k$. Additionally, for all weak embedded UDCR and all leaf children v of b_k, at most three of the six grid positions around v's disk are free.*

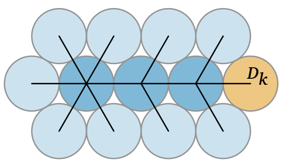
Proof. We show this by induction on the length of the backbone k. For $k = 1$ we have a 6-star which is shown to be rigid in Observation 2.2. From this we also know that all disks in a weak embedded UDCR are placed on the triangular grid. We know that every leaf disk in a weak embedded UDCR of a 6-star has three of its neighboring grid locations already occupied: one by the root node's disk and two by two other leaf nodes' disks. Thus, they all have exactly three grid locations around them left free.

Now, for $k > 1$, let G' be the caterpillar that results from removing all leaves of b_k in G and let Γ' be the resulting embedding. Then G' has backbone $b_1, \ldots, b_{k-1}$ and it remains that $\deg(b_1) = 6$ and $\deg(b_i) = 4$ for all $1 < i \leq k-1$. Since G has a weak embedded UDCR with respect to Γ, G' must also have a weak embedded UDCR with respect to Γ'. We can thus apply the induction hypothesis and it follows that G' is rigid. Let $\mathscr{D}'$ be a weak embedded UDCR for G' obtained by taking a weak UDCR $\mathscr{D}$ for G and removing the disks whose nodes are not in G', i.e., the leaf nodes of b_k. Have a

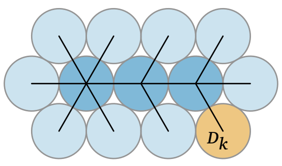
↪



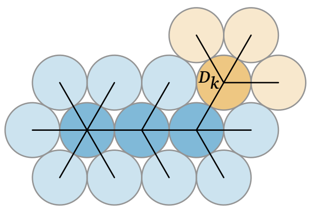
(a) The node b_k is the left child of its parent b_{k-1}.



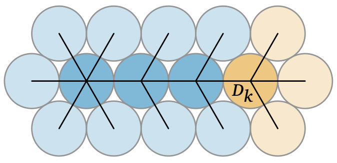
(b) The node b_k is the middle child of its parent b_{k-1}.



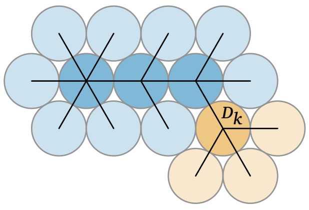
(c) The node b_k is the right child of its parent b_{k-1}.



(d) The resulting weak embedded UDCR of G bends to the left.



(e) The resulting weak embedded UDCR of G continues straight on.



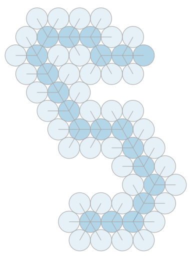
(f) The resulting weak embedded UDCR of G bends to the right.

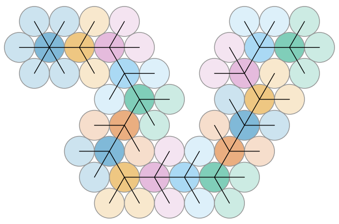
Figure 4.8: An example depiction of the induction step from Lemma 4.3, for $k = 4$.
(a–c) Possible weak embedded UDCRs for the same graph G' with the only difference that b_k (whose disk D_k is highlighted) is the (a) left, (b) middle, or (c) right child of b_{k-1}.
(d–f) The resulting weak embedded UDCRs for the graph G, the three situations corresponding to the three situations in (a–c) above them.

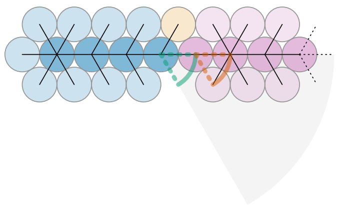
look at Figures 4.8a to 4.8c for three different examples of weak embedded UDCRs for G' for different embeddings of b_k and its sibling leaves relative to b_{k-1} in G'.

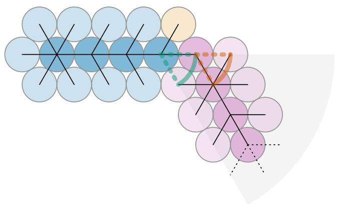
We know, by induction, that $\mathscr{D}'$ is on a triangular grid. From the induction hypothesis it also follows that b_k's disk D_k has at most three grid locations left around it. However, since b_k has three children in G and G has a weak embedded UDCR, there must be space for at least three disks left around D_k. Since Γ tells us in which order the three children of b_k should be drawn, there is exactly one way to place the three child disks of b_k, meaning that G is rigid. Additionally, all three child disks of b_k already have three grid locations around them blocked, as seen in Figures 4.8d to 4.8f where we can see the weak embedded UDCRs for G consistent with the previous weak embedded UDCRs for G'. □

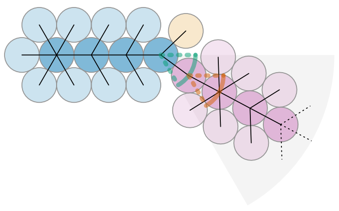
As a result, we are able to construct arbitrary paths that remain on a triangular grid as weak embedded UDCRs of caterpillars. See Figure 4.9a for an example weak embedded UDCRs. Here, the backbone disks are colored differently, together with

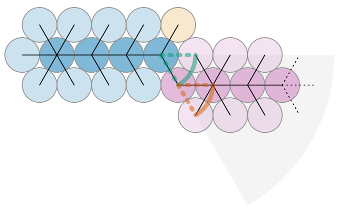



(a) A weak embedded UDCR of a rigid caterpillar.


(b) Both disks are as far to the left as possible.


(c) The first disk is as far to the left as possible while the second is as far to the right as possible.


(d) Both disks that can be moved around are placed somewhere in the middle.


(e) The first disk is as far to the right as possible while the second is as far to the left as possible.

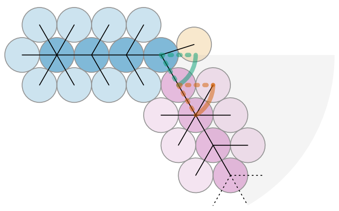
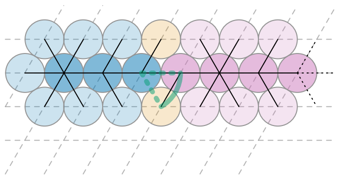
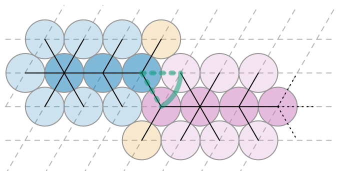

(f) Both disks are as far to the right as possible.

Figure 4.9: Possible weak embedded UDCRs of (a) a rigid caterpillar and (b–f) a caterpillar consisting of two connected rigid parts. In (b–f) we have highlighted the two small regions in which the first and second purple backbone disks can be placed. Furthermore, the bigger gray area with a 60° angle is the area and which the backbone nodes will always stay if the purple caterpillar continues only straight.

their child disks, such that we can distinguish them more easily. We can thus easily see that all but the first and last backbone nodes have exactly two leaf children (degree four). Meanwhile, the first backbone node has five leaf children (degree six) and the last has three leaf children (degree four as well), all exactly as required by Lemma 4.3.

In Figures 4.9b to 4.9f we can see that it is even possible to construct a caterpillar that consists of two rigid parts which are connected in a non-rigid way. Here we have the blue part and the purple part, both of which are rigid (they fulfill the requirements of Lemma 4.3). However, they can rotate relative to each other while being slightly restricted by the orange leaf disk. We can see in the figures that there are two parts that have rotational freedom of 60° each: The first purple backbone disk around the last blue backbone disk and the second purple backbone disk around the first purple backbone disk. This freedom is highlighted by the two (dotted) angles in each figure. Overall, the most extreme representation to the left (Figure 4.9b) and to the right


(a) As far to the top as possible.


(b) As far to the bottom as possible.

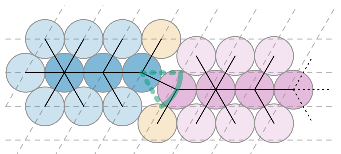

(c) Somewhere between (a) and (b).

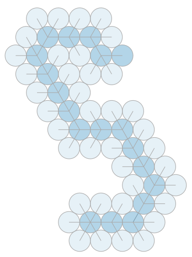
Figure 4.10: Adding another child to the construction from Figures 4.9b to 4.9f at the first purple backbone disk results in less freedom of movement.

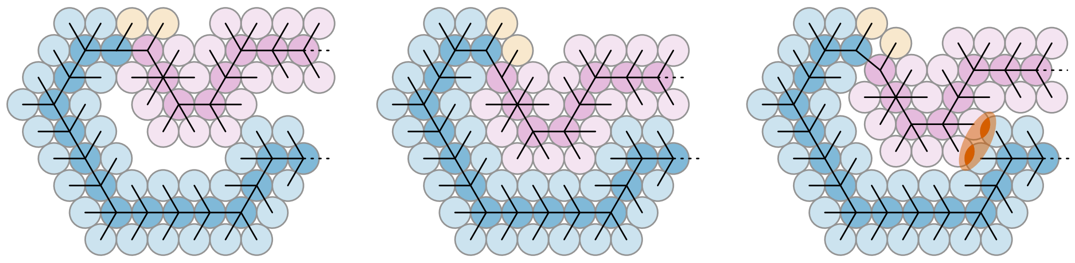
(Figure 4.9f) show that the direction of the purple rigid part will stay inside the gray 60° sector.

Ideally, however, we would like to have more control over the possible weak embedded UDCRs and not have infinitely many possibilities with any angle between 0° and 60°. One way to obtain such a situation can be seen in Figure 4.10. Here we take the caterpillar used in Figures 4.9b to 4.9f and add a leaf disk to the first purple backbone disk. We chose its embedding such that it is placed on the other side of the backbone than the single leaf disk of the previous blue backbone disk. This additional disk prevents the purple weak embedded UDCR from changing its angle relative to the blue weak embedded UDCR. However, the first purple backbone disk can still rotate around the last blue backbone disk with a 60° angle. The resulting weak embedded UDCRs can be grouped into two categories, those that lie completely on the triangular grid and those that do not. In the first category we have two possible weak embedded UDCRs: one in which the purple part is as far to the top as possible, see Figure 4.10a, and one in which it is as far to the bottom as possible, see Figure 4.10b. All other realizations place the purple part somewhere in between and thus not on the grid. An example weak embedded UDCR can be seen in Figure 4.10c.

We have achieved more control over the possible weak embedded UDCRs. However, the number of equivalence classes of the weak embedded UDCRs is still infinite. We would like the weak embedded UDCRs to have only a finite number of equivalence classes and also have that all weak embedded UDCRs are realized on a grid. This is possible by constructing a caterpillar that results in a tight disk packing which prevents movement. See Figure 4.11 for the possible and impossible weak embedded UDCRs of such a caterpillar which we call *two-way gadget*. As we can see in Figures 4.11a and 4.11b, the two extreme positions can be realized without any problems. If, however, we were to try to position the purple weak embedded UDCR in some position in between, it will immediately overlap with the blue weak embedded UDCR on the bottom right, as indicated in Figure 4.11c. It follows that this construction is exactly what we wanted to achieve. A caterpillar with exactly two different weak embedded UDCRs, which

▷ two-way gadget





(a) As far to the top as possible.

(b) As far to the bottom as possible.

(c) An intermediate position forces an overlap.

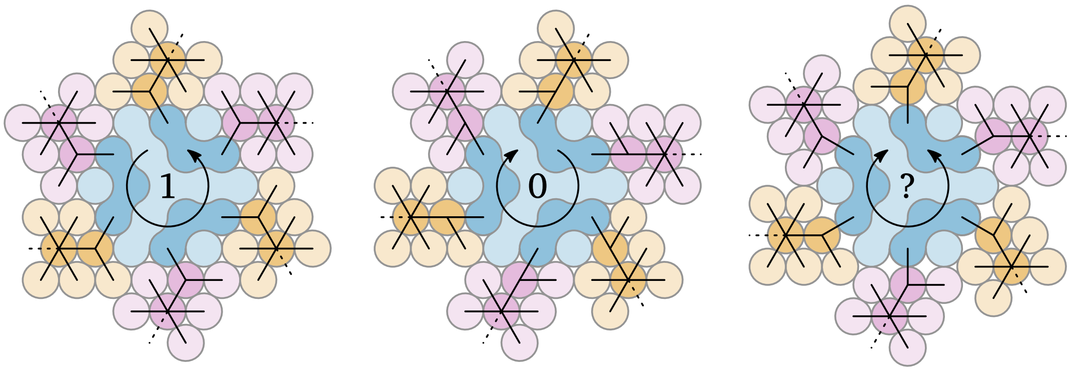
Figure 4.11: The two-way gadget: We use the construction idea from Figure 4.10 and extend the two rigid parts such that they are adjacent on the opposite side of the movement side. This guarantees that any intermediate position, e.g., (c), creates an overlap between the two parts. Then only the two extreme positions (a) and (b) are possible.

both lie completely on a triangular grid. This idea will help us further in the following section.

4.2.3 Variable Gadgets

We now want to combine the ideas developed in the previous section together with the idea of the path that should represent a Planar 3SAT instance, as seen in Figure 4.7b. The goal is to construct a variable gadget, i.e., a caterpillar whose weak embedded UDCRs have exactly two equivalence classes such that they can be assigned the truth values true and false. Furthermore, this gadget must allow for arbitrary many caterpillars going towards a clause gadget (still to be designed) and coming back. The idea for the variable gadget is similar to something we have already seen for the graphs without embedding. In Figure 3.11b in Section 3.2.2 we have seen a small hexagon whose leaf disks can be realized in many ways including two extreme positions on the grid.

Our idea is similar and can be seen in Figure 4.12. Assume that we have a rigid hexagon with radius 2, the light and dark blue parts in our figures. We can now construct six rigid caterpillars (the three purple and three orange weak embedded UDCRs in the figures) that are attached to the disks in the center of each side of the hexagon. Since each caterpillar is connected to the blue hexagon in the same fashion as the construction in Figure 4.10, the caterpillars can be realized in two extreme positions that coincide with the grid (Figures 4.12a and 4.12b) but also all intermediate positions (Figure 4.12c). Furthermore, there are exactly as many disks directly around the blue hexagon as there are grid positions around it (for this see the corresponding arguments in the proof of Lemma 3.4). It follows that placing one such caterpillar also determines the placement of the other caterpillars.

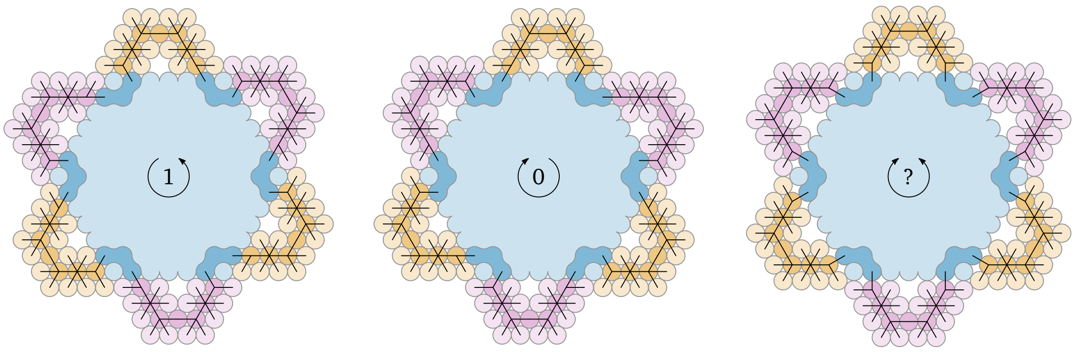


(a) The counterclockwise extreme position.

(b) The clockwise extreme position.

(c) One of the many intermediate positions.

Figure 4.12: Assuming that the blue hexagon in the middle is rigid, the six rigid parts around it can have two extreme positions (a) and (b). However, the weak embedded UDCR can take any position in between, as seen in (c).



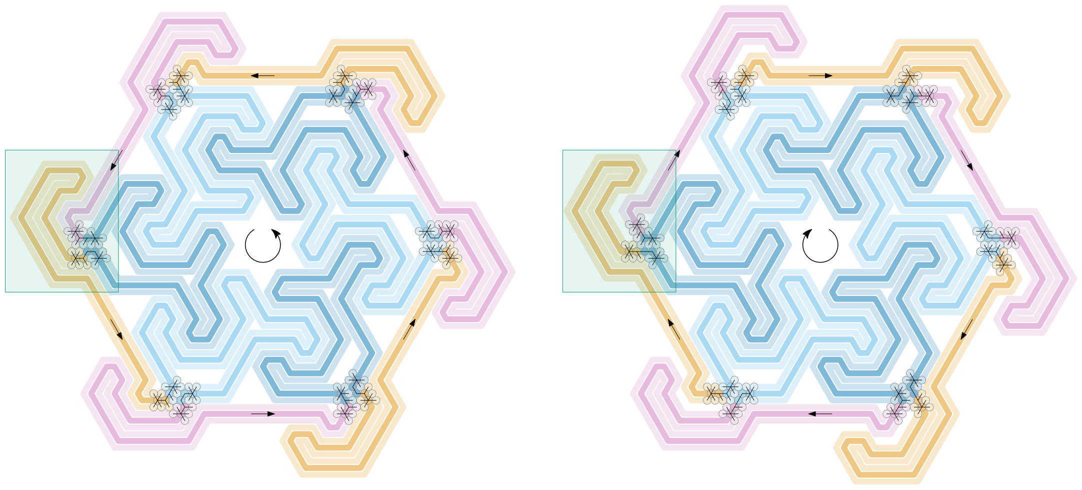
(a)

(b)

(c)

Figure 4.13: A larger version of the construction in Figure 4.12 that still has two extreme positions (a) and (b). However, it remains that infinitely many intermediate positions exist, as seen in (c).

One big assumption in Figure 4.12 is that the inner hexagon is rigid. Furthermore, since the six outgoing caterpillars are not connected to anything, the underlying graph is not a caterpillar. A first step towards mitigating both problems can be found in Figure 4.13. Here, we have grown the inner hexagon and changed the six caterpillars going somewhere undetermined into caterpillars that span the whole side of the hexagon. This keeps the general ideas from Figure 4.12 in place, namely that placing one of the outer caterpillars will determine the position of the other caterpillars. Additionally, the two possible extreme positions from Figures 4.12a and 4.12b can still be found as seen in Figures 4.13a and 4.13b, but it remains that all intermediate positions are possible, e.g., the one in Figure 4.13c. We can also see that the backbone



(a) The outer parts are shifted counterclockwise. The highlight is detailed in Figure 4.15a.

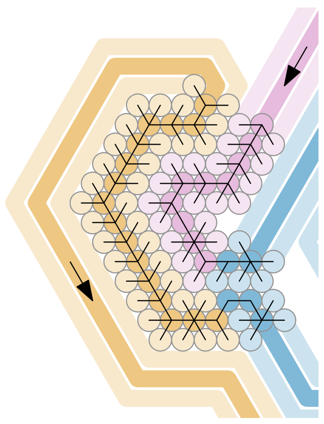
(b) The outer parts are shifted clockwise. The highlight is detailed in Figure 4.15b.

Figure 4.14: The exactly two possible weak embedded UDCRs of the cyclic caterpillar for a variable hexagon. Details of the highlighted parts are found in Figures 4.15a and 4.15b, respectively.

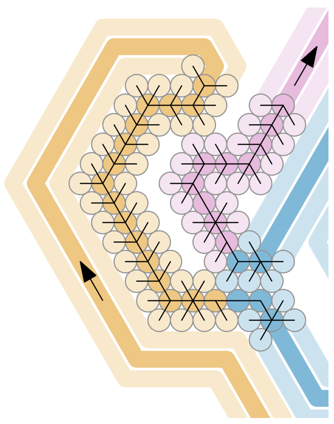
nodes together with the darker nodes of the inner hexagon form a cycle. If we were to cut this cycle open at one position, the resulting graph would be a caterpillar. We call such a graph a *cyclic caterpillar*, which we can also define as any graph that, after removing all leaf nodes once, results in a cycle. In addition to the possible intermediate weak embedded UDCRs, the other remaining problem with Figure 4.13 is that the interior hexagon needs to be made rigid such that the darker blue parts connecting the outer caterpillars, stay in place. These two problems will be mitigated now.

▷ cyclic caterpillar

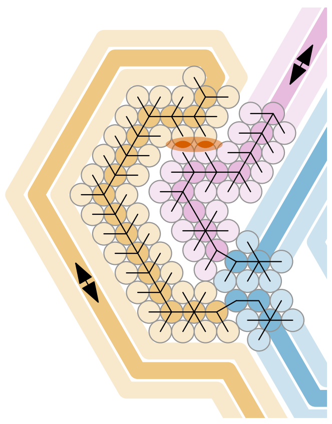
In Figure 4.14 we see the exactly two possible weak embedded UDCRs of the cyclic caterpillar which solves both issues that we just mentioned. The figure uses a more abstract way to represent the weak embedded UDCRs: First, all parts that have the same color are rigid. Second, instead of individual disks, we use line segments to represent the backbone disks (the darker middle color) and the leaf disks (the lighter colors on both sides). Only where two rigid parts meet, we have left the actual disks to see exactly how they connect. We now look at the two issues this construction solves: First, the interior consists of rigid caterpillars whose weak embedded UDCRs interlock with each other and thus prevent them from moving relative to each other. They also provide connections to the outer parts exactly one disk away from each corner, such that moving the outer parts to one extreme position it forces the other parts to move as well. Second, the highlighted part, which can also be seen in more detail in Figure 4.15, uses the two-way gadget from Figure 4.11 to prevent any weak embedded UDCR that



(a) The two outer weak embedded UDCRs are shifted as much counterclockwise as possible.



(b) Both outer weak embedded UDCRs are shifted as much clockwise as possible.



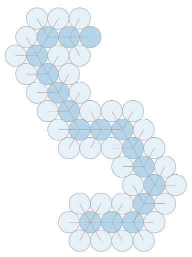
(c) Some disks overlap if the two outer weak embedded UDCRs are not shifted to one of their extremes.

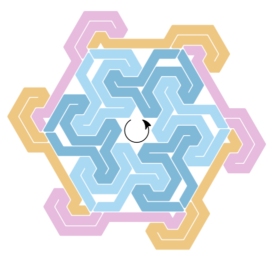
Figure 4.15: One corner of a bigger rigid hexagon around which six caterpillars should move between exactly two possible positions. The possible positions are (a) true and (b) false. (c) Any other position results in an overlap. The colored line segments are used as abstract representations of the weak embedded UDCRs of the caterpillars.

is not one of the two extreme positions. In Figure 4.15c we can see the overlap of the two outer parts if we try to position them outside the extreme positions.

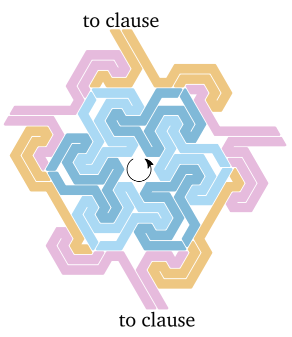
We now have a cyclic caterpillar which has exactly two weak embedded UDCRs. However, looking back at Figure 4.7b we observe that, in order to be used for a variable, we need to be able to extend the caterpillar to the top, right, bottom, and left. To do this, we look at Figure 4.16. Here, the depiction of the weak embedded UDCRs has become even more abstract. Instead of representing a caterpillar's weak embedded UDCR by three adjacent paths with different intensities, we represent it by one thicker path which represents both the backbone and the leaf disks. We can compare Figures 4.14a and 4.16a as well as Figures 4.14b and 4.16d to see that they represent the same weak embedded UDCR, respectively. In order to extend the caterpillar of our construction we can cut open four of the outer parts (top, bottom, top left, and top right) and in each case we run the two (now independent) caterpillars parallel towards one side. The result can be found in Figures 4.16b and 4.16e.

Depending on what happens with the parts to which these two parallel caterpillars will be connected, it may be that these two parts move relative to each other. This needs to be prevented as otherwise we cannot guarantee that the construction works as intended. We thus introduce another gadget to force two adjacent weak embedded UDCRs into a specific position relative to each other. This *lock gadget* can be seen in ▷ lock gadget

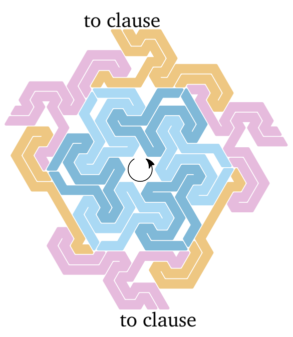




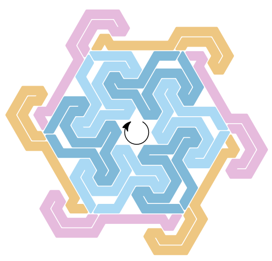
(a)



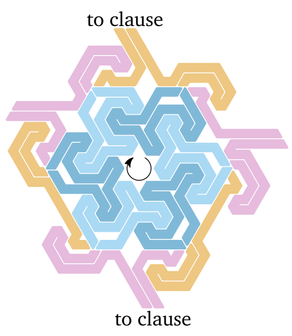
(b)



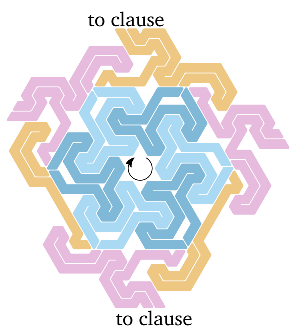
(c)



(d)

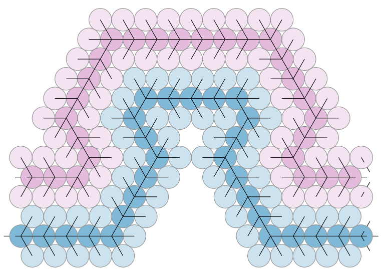


(e)

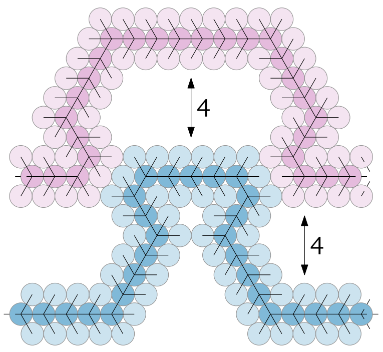


(f)

Figure 4.16: The evolution of the variable hexagon. Starting from the cyclic caterpillar in (a) and (d) we split four outer parts in (b) and (e) and finally add the lock gadget to ensure rigidity of the outer parts in (c) and (f).



(a) The desired weak embedded UDCR of the lock gadget.



(b) To realize both parts apart from each other they must be shifted by four disks.

Figure 4.17: A lock gadget which forces two (mostly) independent caterpillars into a specific weak embedded UDCR relative to each other. It is only possible to find a different weak embedded UDCR if they are apart by at least four disks. By enlarging the construction this distance can be made as big as needed.

Figure 4.17. The idea is that, if the two weak embedded UDCRs can move relative to each other by just a few grid positions, there will always be some overlap unless they are realized exactly as seen in Figure 4.17a. Of course, if they can move further apart, for example four disks, as seen in Figure 4.17b, they can be realized differently. If, however, the possible movement comes just from the two-way gadget (Figure 4.11), i.e., a possible movement of only one disk, then the locked realization is the only possible one. Additionally, if the possible movement is larger but still bounded, we can just grow the lock gadget.

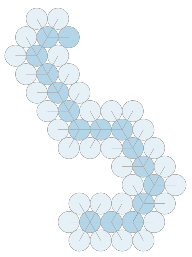
We now take this lock gadget and attach it to the four parallel caterpillars we have constructed in Figures 4.16b and 4.16e and see the resulting weak embedded UDCRs in Figures 4.16c and 4.16f. We call this construction a *variable hexagon*. This now ensures that the two points where these outer parts are connected to the inner parts of the hexagon always move together and thus ensure that shifting one outer part forces all other outer parts to shift as well. The variable hexagon is a caterpillar (or more correctly, a combination of caterpillars) which can be extended on the top, bottom, left, and right and which has exactly two weak embedded UDCRs. However, in order to become a full variable gadget, it needs to be placed next to multiple other variable gadgets which are all realized independently. Additionally, in order to have multiple caterpillars going from a variable gadget to several clause gadgets we will need to join multiple variable hexagons together to form one variable gadget.

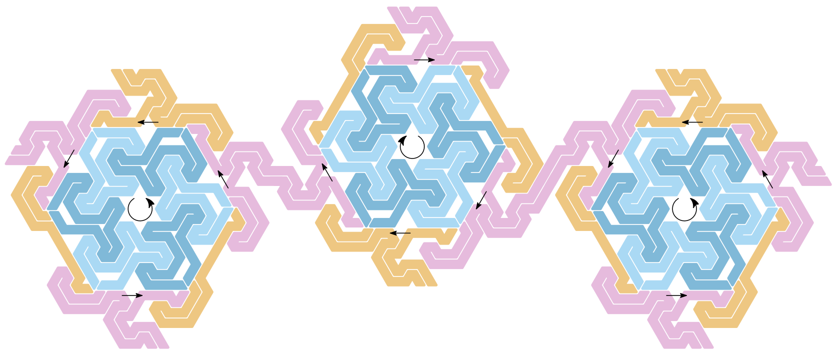
▷ variable hexagon

In Figure 4.18 we show how to construct a *variable gadget* from multiple variable hexagons. We do so by taking a variable hexagon (Figure 4.16) then connecting its two top right outgoing caterpillars to the bottom left outgoing caterpillars of a variable hexagon that has been rotated by 180°. This rotated variable hexagon is then connected via its bottom right outgoing caterpillars to the top left outgoing caterpillars of another (not rotated) variable hexagon. As we can see in Figure 4.18a, the connection between the first and second variable hexagon differs slightly from the connection between the second and third variable hexagon. Since the third variable hexagon is equal to the first we can repeat this construction as often as we want. We can also observe that the outer parts of the variable hexagons rotate in alternating directions, just like gearwheels do. However, this construction works only if we assume the interior of the variable hexagons to be fixed in the plane. Without this, in Figure 4.18a, nothing prevents the interior of the second variable hexagon to move one grid point towards the top left which would make the outer parts turn counterclockwise relative to the interior.

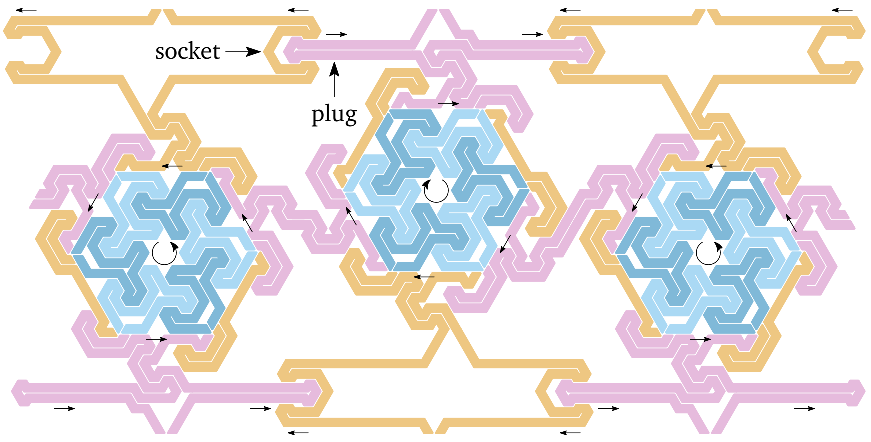
▷ variable gadget

To prevent this from happening we need to fix the interior to its desired position in the plane. We observe that if the variable hexagons' interior parts move, they can never move only to the left or right. There is always a vertical component in its movement when positioning it different from what is desired. Thus, restricting the variable hexagons' vertical movements relative to each other will be effective in preventing any movement of the interiors. We know that if the variable hexagons' interiors do not move, the outer parts on the top and bottom of each variable hexagon only move horizontally

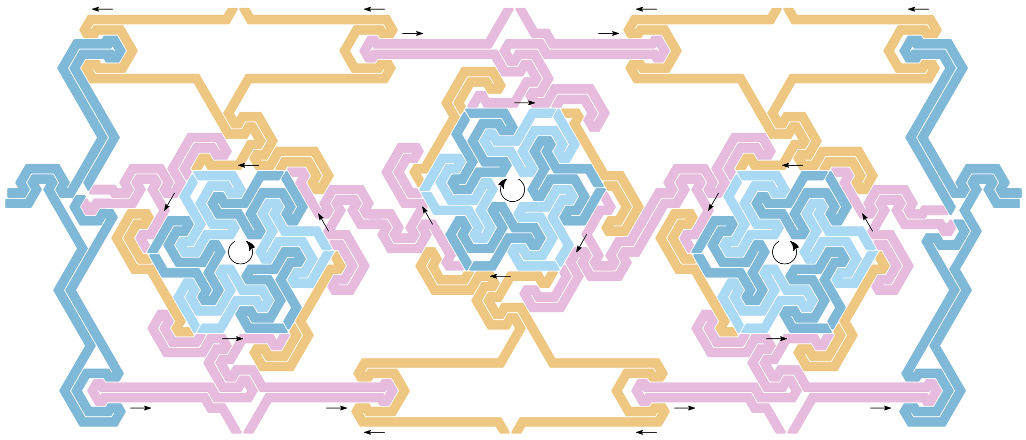




(a) With fixed interior parts, the outer parts' rotational directions alternate like gearwheels.



(b) The plugs and sockets prevent vertical movement, see Figure 4.19.



(c) All (dark and light) blue parts are fixed relative to each other.

Figure 4.18: A variable gadget out of (a) three variable hexagons which can move relative to each other. We can (b) constrain vertical movement due to the interlocking construction and (c) add a fixed frame ensures that all variable hexagons are fixed relative to the frame.

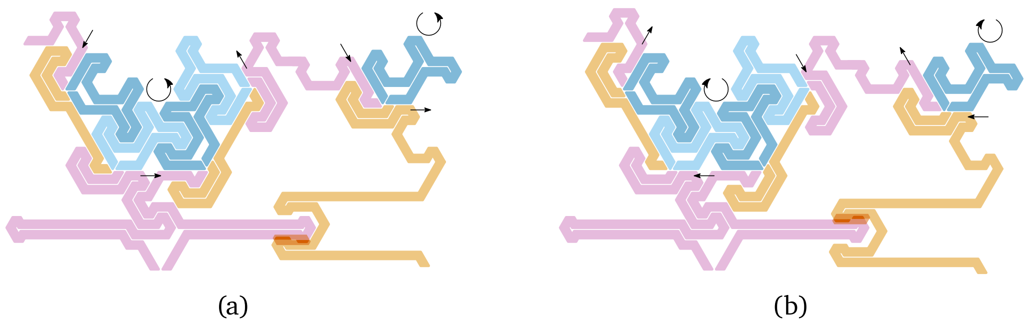


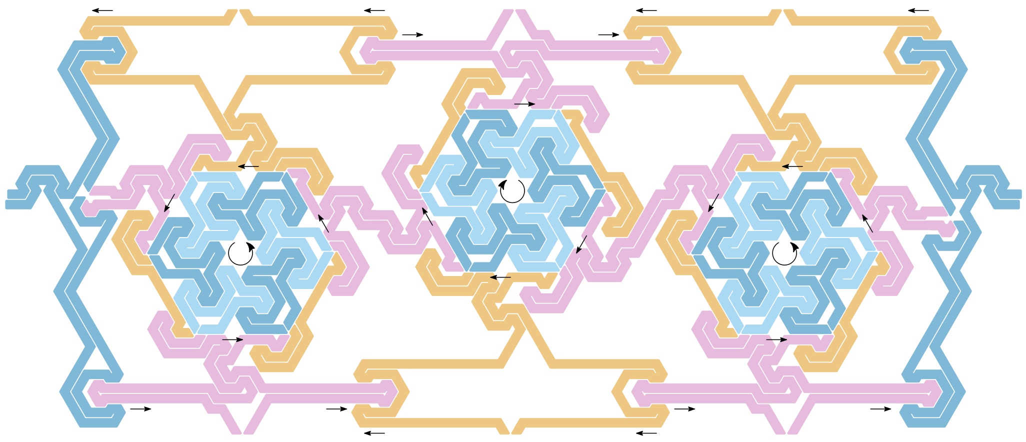
Figure 4.19: Trying to rotate two adjacent variable hexagons in the same direction leads to an overlap of the construction added in Figure 4.18b.

relative to each other. With this information we add more sophisticated caterpillars to the top and bottom (those outer parts that shall connect to the clauses) which prevent any vertical movement relative to each other. The resulting construction can be seen in Figure 4.18b, and we call the two parts the *plug* and *socket*. We can observe that the construction allows the horizontal movement, yet it constrains any (partial) vertical movement. In Figure 4.19 we can see how trying to construct the second variable hexagon in a way that is not consistent with the desired outcome will lead to an overlap between the plug and the socket. More specifically, in Figure 4.19a we have the first variable hexagon in counterclockwise rotation and then try to construct the second one with counterclockwise rotation as well. This leads to the inner parts moving upwards and thus creating the highlighted overlap. The same holds for Figure 4.19b where we try to construct both variable hexagons with clockwise rotation.

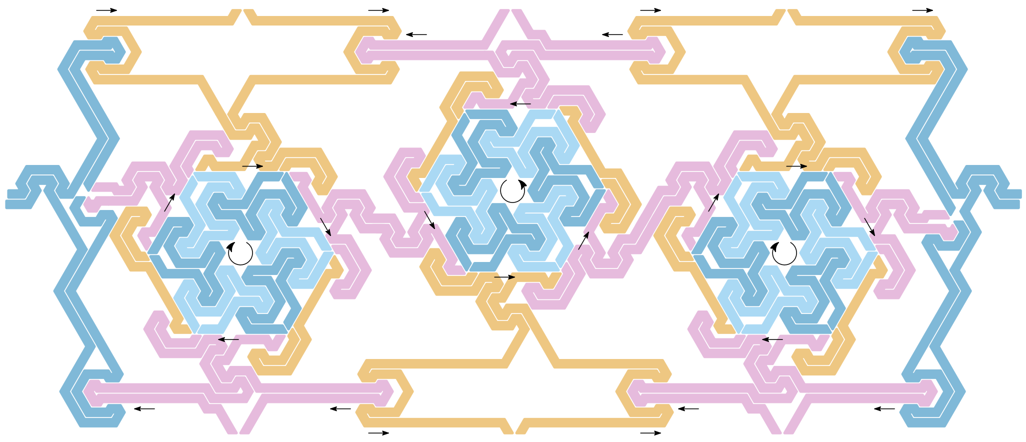
▷ plug

▷ socket

We now have a construction that allows us to chain an arbitrary amount of variable hexagons together such that they turn together like gearwheels in alternating directions. Additionally, the whole construction has exactly two weak embedded UDCRs. What remains open is how to combine multiple such constructions, one for each variable such that their two weak embedded UDCRs can be obtained independently of each other. That is, we want all interior parts of all the variable hexagons for all variables to be in a fixed position relative to each other. The way we do it is by adding a frame to the left and right side of the construction which is kept on the same vertical level by also interfacing with the neighboring variable hexagons through their plugs and sockets. Look at Figure 4.18c where this is shown in blue on the left and right side. Due to the plug and socket the two new parts cannot move vertically with respect to the variable hexagons. Additionally, where the frame is connected to the variable hexagons we use an extended form of the two-way gadget, that can be seen in Figure 4.21. The purple and blue lower part is just the two-way gadget (Figure 4.11) and the purple upper part is fixed relative to the purple lower part due to the lock gadget (Figure 4.17). Both blue parts are also fixed relative to each other by another lock gadget. Thus, in Figure 4.18c, there are exactly two possible ways to position the (orange and purple)



(a) In this realization the purple connectors on the top and bottom are shifted to the right the orange connectors are shifted to the left.



(b) In this realization the purple connectors on the top and bottom are shifted to the left the orange connectors are shifted to the right.

Figure 4.20: A variable gadget consisting of three variable hexagons. In (a) we have the same figure as Figure 4.18c while in (b) we have the only other possibility to realize the underlying graph.

moving parts of the construction relative to the fixed dark and light blue parts. Those two possible ways are found in Figure 4.20. For a breakdown of the order in which the different parts form the whole construction see Figure A.1.

▷ connection points

It is clear that the variable gadget in Figure 4.18c is not a single caterpillar: There are multiple *connection points*, each of which consists of two neighboring caterpillar ends between which an additional caterpillar can or must be inserted. In Figure 4.18c there are eight connection points: one above and one below each of the three variable hexagons, one to the left, and one to the right. If all of these connection points except one are closed, we obtain a single caterpillar. Thus, we can say that we have obtained

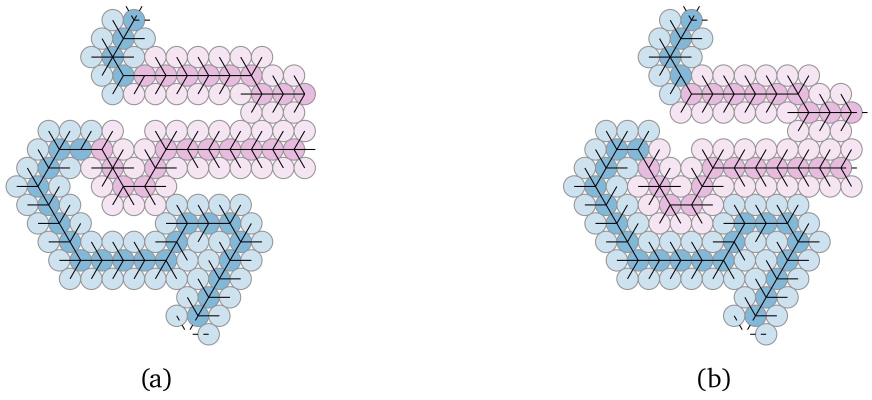


Figure 4.21: The connection between the frame added in Figure 4.18c and the first variable hexagon in the variable gadget. Both possible positions are shown.

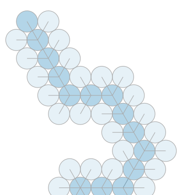
a way to build a caterpillar that has exactly two weak embedded UDCRs and whose number of connection points can be arbitrarily large ($2k + 1$ both on the top and on the bottom for any $k \in \mathbb{N}$). In addition, all connection points above and below the variable gadget move only horizontally and only by exactly one grid point. Both above and below they also alternate their movements going from left to right through all connections points. We can furthermore combine n (for any $n \in \mathbb{N}$) such caterpillars, such that each has two weak embedded UDCRs independent of each other, resulting in a total of 2^n different weak embedded UDCRs overall. This will be used to represent the n variables in a given 3SAT formula.

4.2.4 Clause Gadgets

After seeing how to construct a gadget for each variable we now need to find a gadget for clauses. That is, for every clause C we want to find a construction that is connected to the variable gadgets of the variables that appear in C and that has a weak embedded UDCR if and only if C can be satisfied.

Similar to other clause gadgets in Planar 3SAT reductions (e.g., [Bow+15]) we take the idea that a clause gadget for a clause C is a finite space that is partly occupied by each literal that has a truth value false. If we can then constrain the space for a clause such that it can only accommodate at most two literals, we know that at least one literal must be true. In our case we will design the gadget such that each literal has a weak embedded UDCR which will want to occupy the clause space in case the literal is false; however, at most two of the three weak embedded UDCRs can occupy the space.

In Figure 4.22 we show the idea of the gadget that we will design. Here, the space to be occupied by at most two of the three parts is the dashed central line. In Figure 4.22a all three parts are in their configuration for true and thus none of them has a disk on a



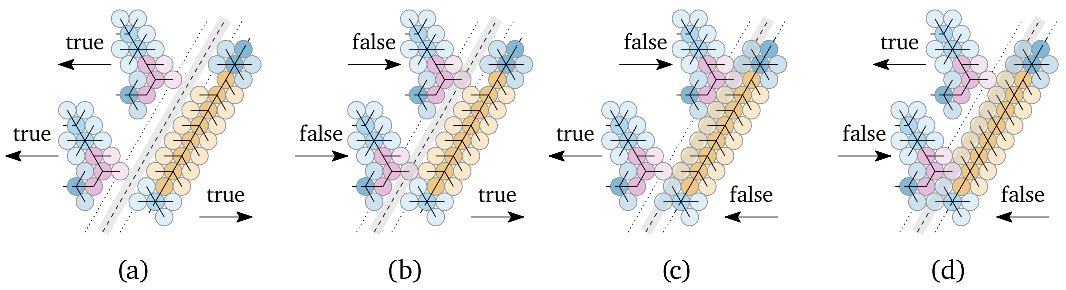
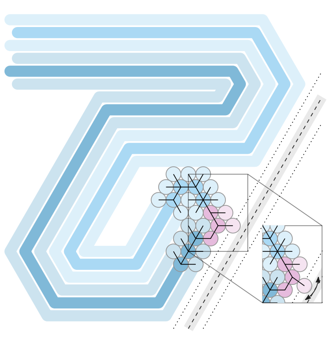


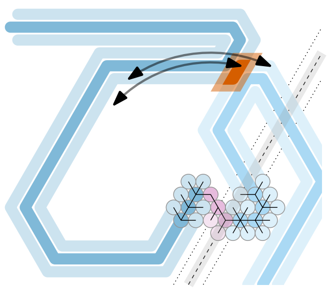
Figure 4.22: The idea for the clause gadget. A false literal will occupy space on the central dashed line. At most two of the three parts can occupy space there at the same time.

grid position on the line. If any of the three parts is in its configuration for false it is moved exactly one grid point horizontally towards the center line. The two left parts are moved to the right and thus one disk occupies one grid point on the central line. The right part is moved to the left and thus between the two fixed parts on the top and bottom, it occupied all but one position on the center line. It follows that it is impossible to move all three parts towards the center line as this would occupy one more grid point than there is available. In Figures 4.22b to 4.22d we can see the three possible representations in which exactly one part is in its true configuration.

Of course, the weak embedded UDCRs shown in Figure 4.22 are not rigid, especially the two on the left. However, in Figures 4.23 and 4.24 we show two caterpillars, one for the left and one for the right part, that are rigid except for the purple parts. Both constructions are shown in their true configuration with respect to the shown center line. For the left part we can see in Figures 4.23b and 4.23c two examples where we try to place the purple backbone disks differently from what we can see in Figure 4.23a. In Figure 4.23b we see the most extreme clockwise rotation possible, but we observe that there is an overlap in the top right part. An attempt to not turn it as far can be seen in Figure 4.23c which also has an overlap. The arrows indicate the movement of two disks between the two figures. As we can see, there is always at least one disk that intersects with the darker caterpillar's weak embedded UDCR. Thus, the only part in Figure 4.23a that can move arbitrarily is the lower purple leaf disk which can be placed anywhere on a 60° degree arc. However, no matter where it is placed along the arc, its center will be on or on the other side of the center line, if the construction is realized in its false configuration. It is also the only disk of the whole construction to be on this line.

The same idea holds true for the right part which is detailed in Figure 4.24. The purple part is not rigid and it may bend by 60° as there is one leaf disk missing. However, as exemplified in Figures 4.24b and 4.24c if we try to bend the purple part we will have an intersection of the blue parts. Thus, the weak embedded UDCR in Figure 4.24a is the only possible way to realize the two blue parts relative to each other. However,


(a) The only weak embedded UDCRs; the one purple disk can rotate a bit.


(b) Due to the overlap, this is not a valid weak embedded UDCR.

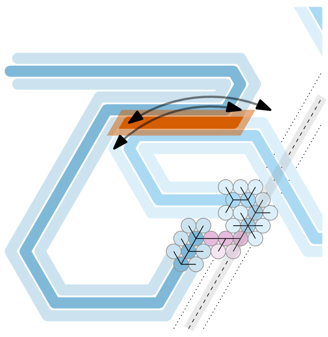
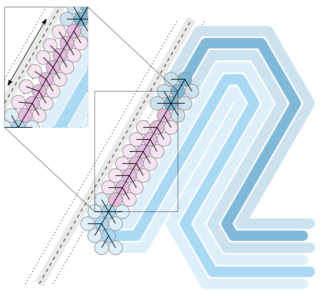
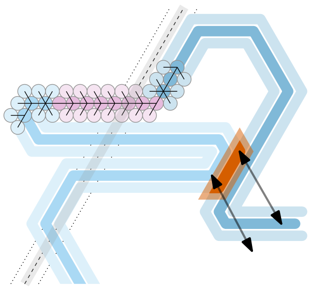

(c) Another impossible weak embedded UDCR.

Figure 4.23: A caterpillar for the left parts of a clause gadget. Apart from the movement of one disk, the rest of the caterpillar is rigid. As indicated in (b) and (c), any rotation has an overlap.


(a) The only weak embedded UDCRs; the purple disks can move around a bit.


(b) Due to the overlap, this is not a valid weak embedded UDCR.

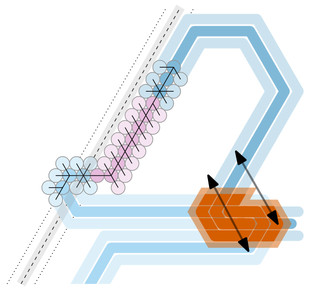

(c) Another impossible weak embedded UDCR.

Figure 4.24: A caterpillar for the right part of a clause gadget. The purple disks on the left can move around a bit, while the rest of the caterpillar is rigid.

due to the lack of one leaf disk, the other purple leaf disks on the left side can move around on an at most 60° arc, depending on the placement of the other leaf disks.

Using the two constructions with the same idea as in Figure 4.22 we can see one possible weak embedded UDCR of a clause gadget in Figure 4.25. The construction from Figure 4.24 is increased in length such that both left parts can fit into the one disk space left by this construction. In the figure we can see that two of the three parts are in their false configuration and one (bottom left) is in its true configuration. It is clear that trying to put it in its false configuration as well, i.e., moving it one grid point to the right, is not possible.

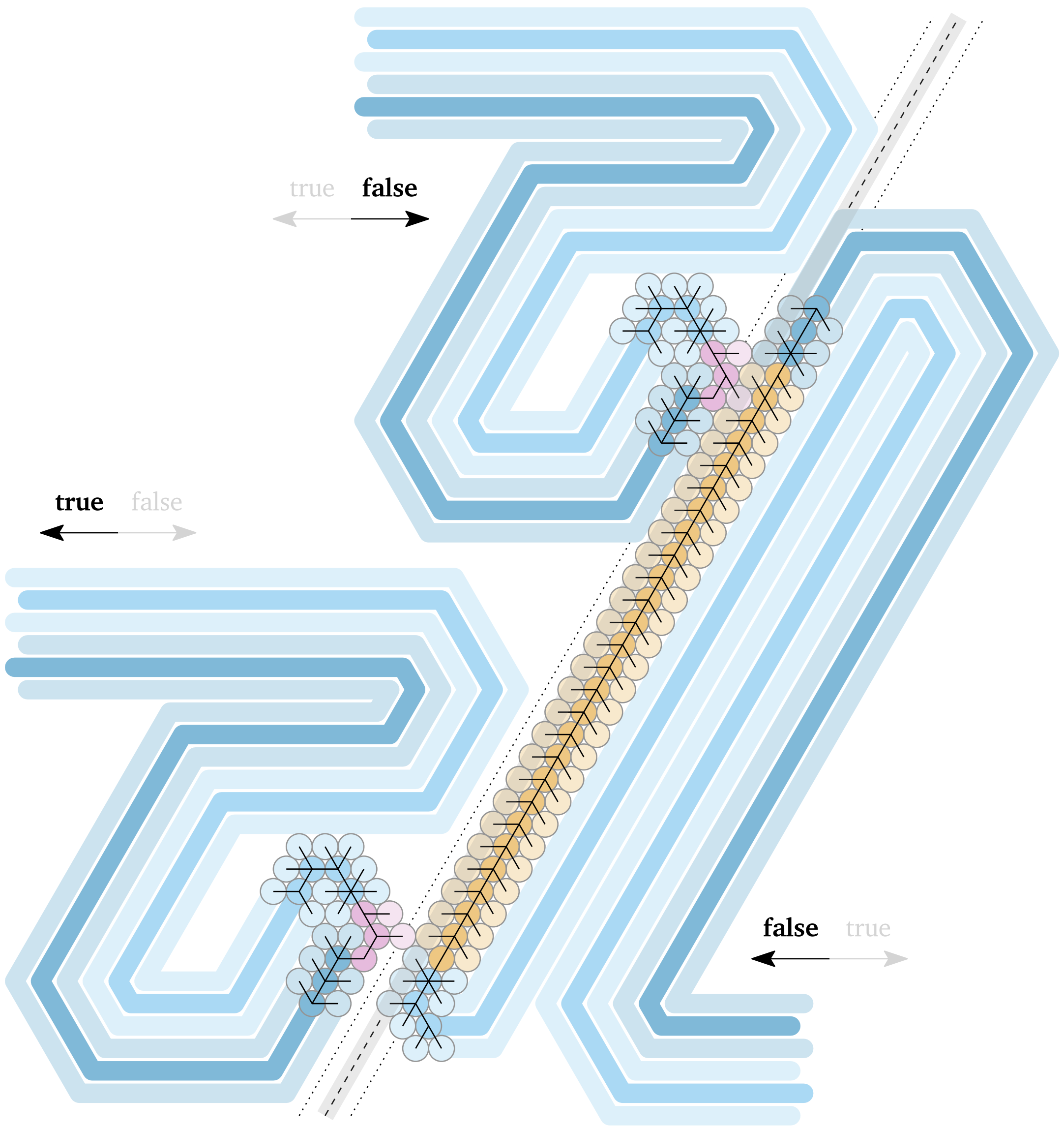


Figure 4.25: One possible weak embedded UDCR of the full clause gadget where two literals (top left and right) are set to false and the third one (bottom left) is set to true.

4.2.5 Putting Everything Together

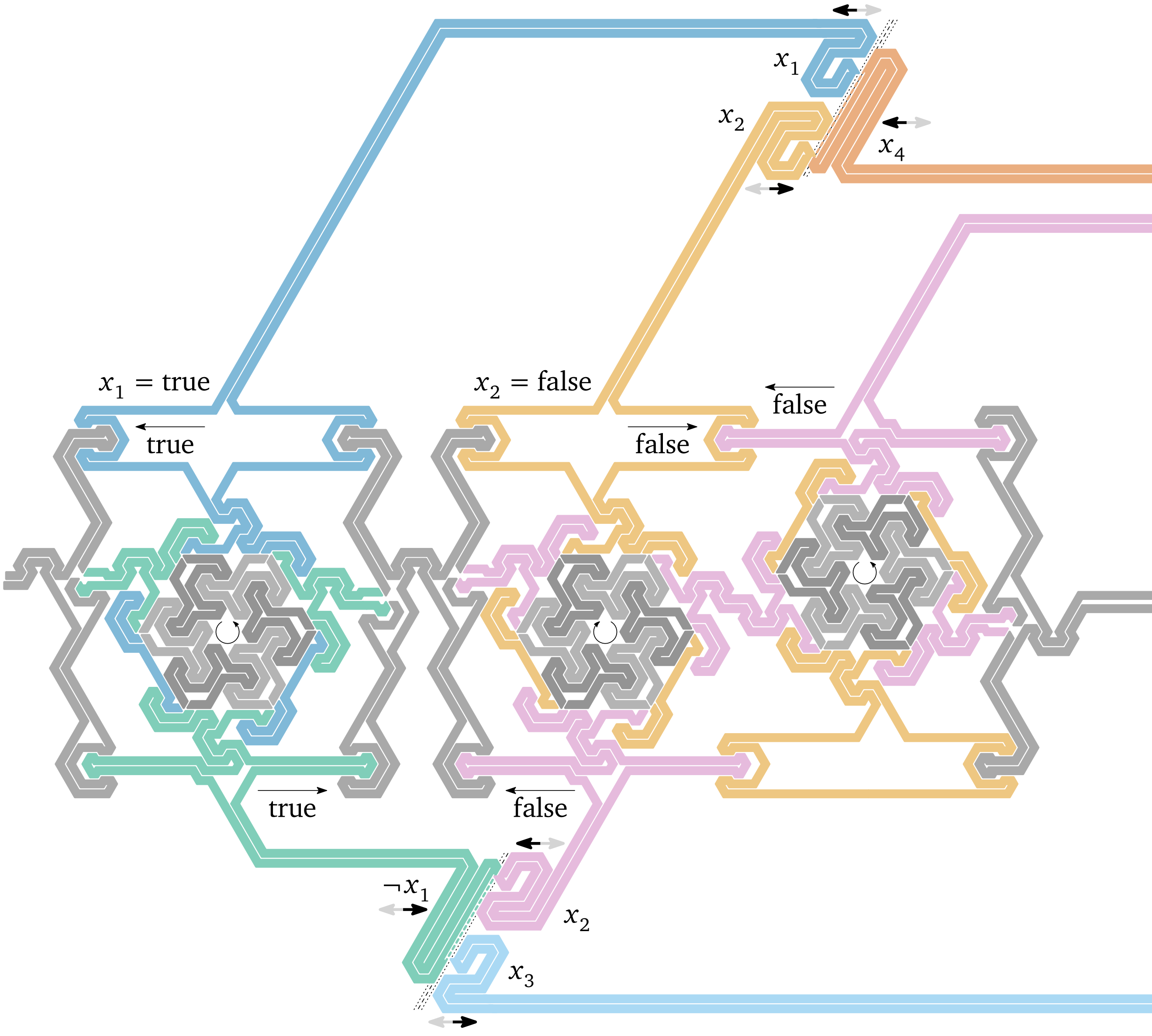
With the variable and clause gadgets we look back at the full picture for a given formula, as exemplified in Figure 4.7. An observant reader may have wondered why, during the construction of the variable gadget, we have never specified which of the two possible weak embedded UDCRs should mean true and which should mean false. The reason is that we can choose this for every variable gadget independently, as long as it is consistent with all clause gadgets it is connected to. This brings us to the last remaining task: connecting the variable gadgets to the clause gadgets. A full picture of one weak embedded UDCR of the caterpillar for the example formula $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$ (the one also used previously) can be found in Figure 4.26 (pages 74 and 75). Note that, in contrast to the variable gadgets we have seen before (e.g., in Figure 4.18) the variable gadgets used in this figure only consist of one or two variable hexagons. This was an optimization we made to fit the whole weak embedded UDCR into a readable figure. We will make some additional remarks regarding this optimization in a few paragraphs. For the construction of the caterpillar we had to decide for every variable which of its two weak embedded UDCRs should have the meaning of true and which the meaning of false.

For x_1 we have two clauses: c_1 on the bottom in which x_1 appears as a negative literal $\neg x_1$ and c_2 on the top in which x_1 appears as x_1. For both clause gadgets the extensions of the variable gadget for x_1 enter the clause gadget from the left. A false value for the literal means that the respective part of the weak embedded UDCR pushes into the clause gadget, which means moving to the right in these two cases. Since one literal is the positive literal x_1 and one literal is the negative literal $\neg x_1$ we only need one variable hexagon in the variable gadget for x_1: We define a counterclockwise rotation to mean true and a clockwise rotation to mean false. Then, setting x_1 to true moves the literal x_1 out of the clause gadget for c_2 on the top and pushes the literal $\neg x_1$ into the clause gadget for c_1 on the bottom. Setting x_1 to false does the exact opposite, exactly as we need.

Then the top and bottom can be directly connected to their respective clause gadgets, and we only need one variable hexagon in the variable gadget.

For x_2 we need to connect to all three clause gadgets. Additionally, in the two gadgets on top x_2 appears once as x_2 and once as $\neg x_2$. Thus, we use two variable hexagons for x_2 such that for c_2 setting x_2 to true means moving to the left where for c_3 setting x_2 to false means moving to the left. The last clause gadget to which we need to connect x_2 is c_1 on the bottom in which x_2 appears as x_2. We locate it to the left of x_2 and thus from the two possible connection points on the bottom we choose the one for which setting x_2 to true means moving to the right.

For x_3 and x_4 we can apply the same ideas and see that for both we also only need one variable hexagon: In both clauses it appears in, x_3 appears as x_3, on the top setting it to true should move to the left and on the bottom it should move to the right. For x_4


x_1
x_2
x_4
x_1 = true
true
x_2 = false
false
false
true
false
$\neg x_1$
x_2
x_3

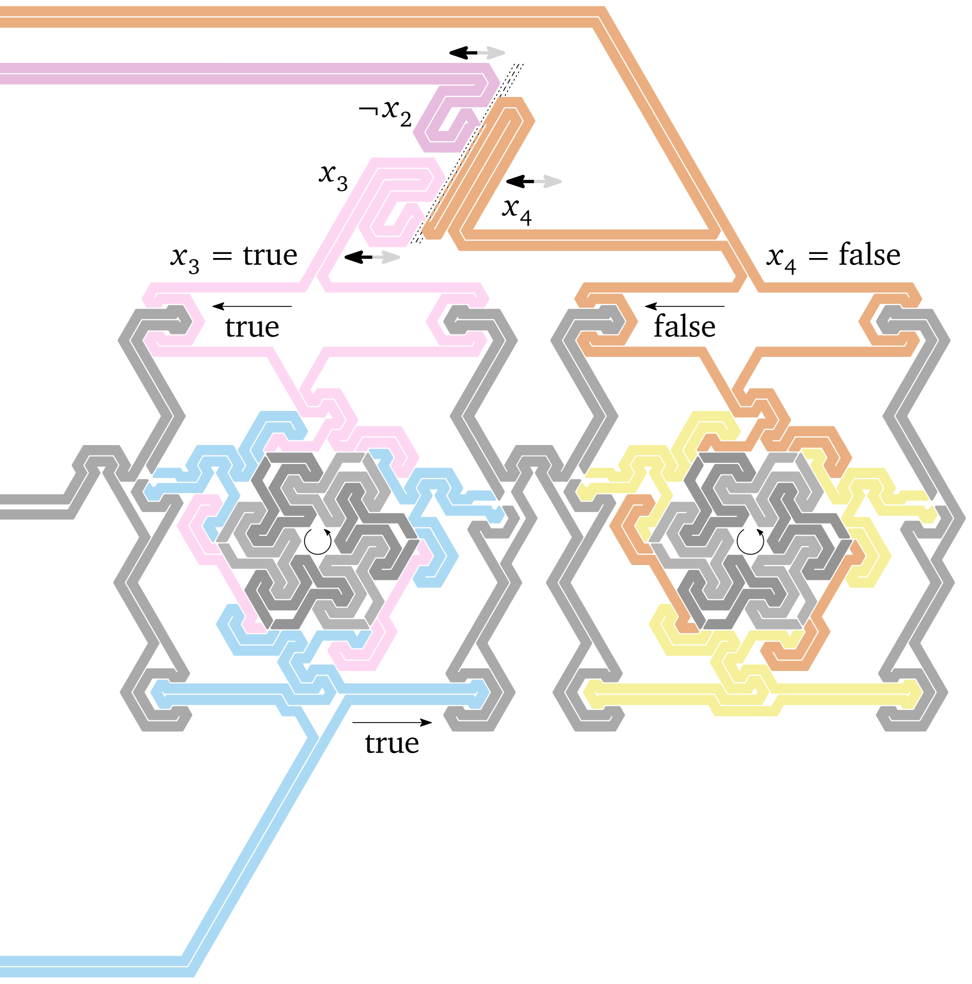


Figure 4.26: One full weak embedded UDCR for the caterpillar constructed for the 3SAT formula $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$. The construction implies the choice of x_1 = true, x_2 = false, x_3 = true, and x_4 = false which is a satisfying assignment for ϕ.

we only have two clauses, both on the top, and it appears as x_4 in both. We can thus connect both clauses to the same connection point as one caterpillar.

From the realization in Figure 4.26 we can then infer that setting x_1 and x_3 to true as well as x_2 and x_4 to false is a valid assignment for ϕ. This can be easily checked to be true as $x_3 =$ true satisfies $c_1 = (\neg x_1 \lor x_2 \lor x_3)$ and $c_3 = (\neg x_2 \lor x_3 \lor x_4)$ while $x_1 =$ true satisfies $c_2 = (x_1 \lor x_2 \lor x_4)$.

As noted before, in Figure 4.26 some choices were made such that we need as few variable hexagons for each variable as possible. The reason is that the figure should not be too large and thus unreadable in the details. However, we could also choose a less creative and more rigid way of building the caterpillar. Given a rectilinear drawing of the planar embedding of the associated graph $G(\phi)$ of a 3SAT formula ϕ, let t_i and b_i be the number of clauses x_i is connected on the top and on the bottom, respectively. For the variable gadget for x_i we could then use $2 \cdot \max\{t_i, b_i\}$ variable hexagons. This way we could always say that a counterclockwise rotation of the leftmost variable hexagon means that the variable is set to true and a clockwise rotation means that it is set to false. We would have sufficient connection points to connect all relevant clause gadgets to the variable as at least two connection points with opposite movements are reserved for each clause gadget.

Overall, from the construction we can conclude

Theorem 4.2. *The problem of finding a weak embedded UDCR for a caterpillar with a given embedding is NP-hard.*

Proof. As seen in the whole Section 4.2, we reduce from Planar 3SAT. Given a 3SAT formula ϕ with n variables and m clauses we find a rectilinear drawing of a planar embedding of its associated graph. From this drawing we then see how many variable hexagons we need for each variable and how to connect the clause gadgets to the variable gadgets, such that we can construct the caterpillar $C(\phi)$ for ϕ.

To ensure that we can construct $C(\phi)$ in polynomial time, we bound its size by looking at the individual components:

- We have m clause gadgets each of which has constant size. Thus, we need $O(m)$ nodes for all clause gadgets.
- We have n variable gadgets which in total need at most $6m$ variable hexagons: at most 2 for each of the $3m$ literals in the formula. Each variable hexagon has constant size which means that all variable gadgets need $O(n + m)$ nodes for all variable gadgets.
- What is left are the caterpillar parts which connect the variable gadgets to the clause gadgets. Each such connection goes vertically by at most the number of clauses and horizontally by at most the number of variable hexagons plus the

↪

number of variable gadgets. This leads to $O(m+n)$ necessary nodes for each connection, totaling in $O(m^2+mn)$ nodes for all connections.

Overall, the size of $C(\phi)$ is bounded by $O(m^2+mn)$.

We can also see that $C(\phi)$ has a weak embedded UDCR if and only if ϕ is satisfiable. If it is satisfiable we have a satisfying variable assignment which means that in the weak embedded UDCR in which the variable gadgets resemble this assignment, for every clause gadget we have at least one of the three parts that is not pushed into the gadget, making it realizable. On the other hand, if we find a weak embedded UDCR we know that every variable gadget is in exactly one of two states which has the meaning of setting the corresponding variable to true or false. In the weak embedded UDCR each clause gadget must have at least one part that is not pushed into the gadget. This part corresponds to the literal that satisfies this clause. □

Similar to the previous chapter for graphs without embedding, the obvious question now is whether the problem is not only NP-hard but also contained in NP and thus NP-complete. As before, we do not have an answer to this question for the general problem, but we can again show that the problem is contained in $\exists\mathbb{R}$.

Given a graph, introduce one pair of variables x_v and y_v for each node v. For each pair of nodes u, v such that u and v are connected by an edge we add the constraint $(x_u - x_v)^2 + (y_u - y_v)^2 = 2^2$ to ensure that their disks touch. For all other pairs u, v we add $(x_u - x_v)^2 + (y_u - y_v)^2 \geq 2^2$ to ensure that no two disks intersect. So far this is the same construction as for graphs without embedding, and it will give as a weak UDCR if possible. However, this will generally not conform to the given embedding.

Given the weak UDCR we can easily check whether it conforms to the embedding with the following test: We draw an oriented line through the centers of two disks that should be next to each other according to the embedding. The orientation of the line is from the first to the second disk in counterclockwise order. Now, there is a disk in between the first and the second in counterclockwise order if and only if this disk's center is to the right of the oriented line. Similarly, there is a disk between the second and the first in counterclockwise order if and only if this disk's center is to the left of the oriented line. See Figure 4.27 for two situations, the first in which all other disks are to the left of the line and the second where there is a disk to the right of the line. We thus only need to add constraints in the following way for every node with more than two adjacent nodes. We look at all adjacent nodes and their cyclic order. For every successive pair of nodes u, v in counterclockwise order and every remaining node w we add the constraint that (x_w, y_w) must be to the left of the oriented line through (x_u, y_u) and (x_v, y_v). This constraint can be easily checked with the orientation test, see, e.g., Preparata and Shamos [PS93, Section 2.2.1] or de Berg, Cheong, van Kreveld, and Overmars [Ber+08, Excercise 1.4.a]. The corresponding constraint added for each such node triple is then

$$x_u y_v - x_u y_w - x_v y_u + x_v y_w + x_w y_u - x_w y_v > 0.$$

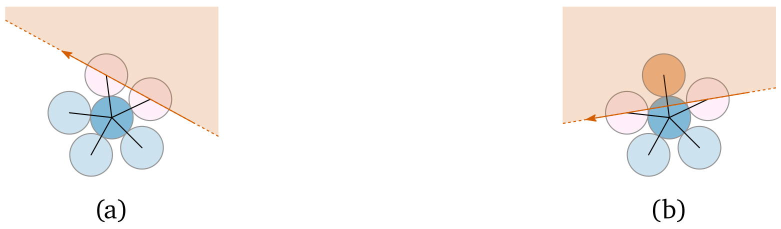


Figure 4.27: (a) If all disk centers of the other child disks are to the left of an oriented line through two disk centers, we know that those two disks are next to each other in the embedding. (b) If, on the other hand, a disk center is to the right of this oriented line, we know that this disk is placed between those two disks.

Since every node has at most six adjacent nodes (otherwise it cannot have a weak embedded UDCR) the number of these additional constraints is linear in the total number of nodes. Thus, the constructed formula still has polynomial size and can be constructed in polynomial time. We have also argued that these additional constraints exactly model the additional constraints due to the embedding. This shows that the problem is contained in $\exists\mathbb{R}$.

If, however, we are only interested in finding a grid-restricted weak UDCR, we can show that the problem is contained in NP. Given a graph the certificate of a grid-restricted weak UDCR gives us the disk center for each node. Since the disks centers are all on the hexagonal grid, their coordinates can be given as integers. The verification algorithm then only needs to check that no two disks are placed on the same grid position and that the disks of adjacent nodes are placed on adjacent grid positions. This can be easily done in quadratic time by checking all pairs of nodes. In addition, for each node we need to check whether the placement of the disks of its adjacent nodes conforms to the given embedding. Since there are only six possible positions and up to six adjacent nodes, this can be done in constant time per node. This approach does not work if the weak UDCRs are not grid-restricted, since we cannot be sure that the disk coordinates have polynomial size in the input.

We can now conclude this chapter with a summary of our results. In Section 4.1 we showed a linear-time algorithm for finding a grid-restricted weak embedded UDCR for strictly x-monotone caterpillars with a given embedding. The general problem of finding a weak embedded UDCR is NP-hard for caterpillars, as shown in Section 4.2, by giving a reduction from Planar 3SAT. We showed that the problem is in $\exists\mathbb{R}$ and in the grid-restricted case it is even NP-complete.

II

Colored Nearest Neighbor Graphs

Chapter 5

Introduction and Preliminaries

In the well-known *connect the dots* puzzle [Wik23], a form of puzzle generally created for children, we are given n points in the plane, numbered from 1 to n. The task is then to connect two successively numbered points with a straight line segment. The result is a straight-line drawing of a path. This type of puzzle dates back at least over a hundred years as an example in a newspaper from 1915, seen in Figure 5.1, shows [She15].

A somewhat similar puzzle was introduced by van Kapel [Kap14] ten years ago, where we are given a set of points in the plane, each of which has at least one color. The task is then as follows: for each point and each color it has, we look for the closest point that also has this color and connect both points with a straight line segment. Since this problem is easy to solve, we are more interested in the problem of creating such a puzzle. Given a set of line segments, how do we color their endpoints such that connecting them as described before results in the same set of line segments?

A slightly different variant of the problem where each point is given *exactly* one color is shown to be NP-complete for two dimensions and more than two colors by Cleve et al. [Cle+22]. If maximal line segments are allowed to have points in their interior then it even remains NP-complete for two colors. As a result, we will focus on the one-dimensional case, where we are given a sorted set of one-dimensional points. We want to color the points such that any resulting connection is only between neighboring points, and no connection is made more than once.

The organization of this chapter is as follows. In Section 5.1 we first define multisets and multigraphs, which is used afterwards in Section 5.2 to define what a (colored) nearest neighbor graph is, in general. Section 5.3 then goes into more details about the one-dimensional case, defines the computational problems we will talk about, and make some useful observations about the one-dimensional case.

5.1 Multisets and Multigraphs

We will need the notion of multisets and multigraphs later to properly define colored nearest neighbor graphs. Thus, we will first introduce some basic definitions for multisets, based on Syropoulos [Syr01].

FRIDAY, DECEMBER 3, 1915.
THE GREAT DOT MYSTERY
When Freddy awoke he knew that his dream never would come true because his mother disliked birds. So he put the matter up to the cook. "Do you know what my birthday present is to be?" he asked. "Maybe I do and maybe I don't," she replied. "maybe its going to be a ———
TO BE CONTINUED.

Figure 5.1: A newspaper clipping from December 1915 showing a *connect the dots* puzzle [She15].

▷ multiset
▷ multiplicity
▷ multiset equality

Definition 5.1 (cf. [Syr01, Definitions 2 and 4]). A *multiset* $\mathbf{S}$ over a set U is a pair $(U, m_{\mathbf{S}})$, where $m_{\mathbf{S}} : U \to \mathbb{N}$ is a function that tells us the *multiplicity* of each element of U in $\mathbf{S}$.

Two multisets $\mathbf{S} = (U, m_{\mathbf{S}})$ and $\mathbf{T} = (V, m_{\mathbf{T}})$ are equal, written as $\mathbf{S} = \mathbf{T}$, if and only if $U = V$ and for all $x \in U$ we have $m_{\mathbf{S}}(x) = m_{\mathbf{T}}(x)$.

With this definition, it is easy to see that ordinary sets are just special cases of multisets:

Observation 5.1 (cf. [Syr01, Remark 1]). *Any set $S \subseteq U$ that is a subset of a universe U is a multiset (U, χ_S) where $\chi_S : U \to \{0, 1\}$ is the characteristic function of S, that is,*

$$\chi_S(x) = \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{otherwise.} \end{cases}$$

↪

We can thus compare the equality of a multiset $\mathbf{S}$ *and a set* S *by checking whether* $\mathbf{S} = (U, \chi_S)$ *or not. Similarly, every multiset in which each element appears either exactly once or not at all can be considered a set.* □

Due to the previous observation, we can interpret any set as a multiset and any multiset that contains each element at most once as a set. If an element appears in two multisets, we want an operation that combines both multisets by adding the individual multiplicities of the element:

Definition 5.2 (cf. [Syr01, Definition 7]). Given two multisets $\mathbf{A} = (U, m_{\mathbf{A}})$ and $\mathbf{B} = (U, m_{\mathbf{B}})$ over the same set U we define the *multiset sum* $\mathbf{A} \uplus \mathbf{B}$ of $\mathbf{A}$ and $\mathbf{B}$ as the multiset $\mathbf{C} = (U, m_{\mathbf{C}})$ with $m_{\mathbf{C}}(x) = m_{\mathbf{A}}(x) + m_{\mathbf{B}}(x)$ for all $x \in U$.

▷ multiset sum

Similar to sets and multisets, we have the notion of graphs and multigraphs. While graphs only have a binary value that decides for each edge whether it is in the graph or not, a multigraph can have the same edge multiple times:

Definition 5.3. An undirected *multigraph* G is a pair $(V, \mathbf{E})$ where V can be any set of nodes and $\mathbf{E} = \left(\binom{V}{2}, m_{\mathbf{E}}\right)$ is a multiset of undirected edges.

▷ multigraph

Similarly to multisets, an undirected multigraph $G = (V, \mathbf{E})$ where the edge multiset $\mathbf{E}$ contains each edge at most once can and will be interpreted in this thesis as an undirected graph G with the edge set $\mathbf{E}$ interpreted as a set.

5.2 Defining Colored Nearest Neighbor Graphs

Even though our focus will be on the one-dimensional case, we will first define the different aspects that lead to the more generally defined problem.

5.2.1 Nearest Neighbor Graphs

We first begin by defining what it means to be a nearest neighbor.

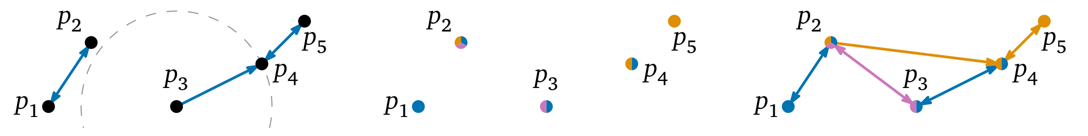
Definition 5.4. Let $P \subseteq \mathbb{R}^d$ be a point set in the d-dimensional Euclidean space and let $p, q \in P$ such that $p \neq q$. Then q is the *unique nearest neighbor* of p, denoted as $p \multimap q$, if and only if

$$\operatorname{dist}(p, q) < \min_{r \in P \setminus \{p,q\}} \operatorname{dist}(p, r)$$

where $\operatorname{dist} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{\geq 0}$ is the Euclidean distance metric. Thus, $\multimap$ is a binary relation on the point set, i.e., $\multimap : P \times P$. We also say that *p connects to q*.

▷ unique nearest neighbor
▷ connects to

With this definition, we have two immediate observations:



(a) An NNG of five points P where the circle indicates $p_3 \multimap p_4$. As $p_4 \not\multimap p_3$, $\multimap$ is not symmetric.

(b) This color assignment σ assigns $\{1\}$ to p_1, $\{1, 2, 3\}$ to p_2, $\{1, 2\}$ to p_3, $\{1, 3\}$ to p_4, and $\{3\}$ to p_5.

(c) In $\mathcal{N}(P, \sigma)$ for σ no edge appears in more than one color; it is thus a CNNG.

Figure 5.2: An example point set with an NNG, a color assignment, and a CNNG.

Observation 5.2. *The relation $\multimap$ is generally not symmetric. Furthermore, $\multimap$ is a partial function $\multimap : P \to P$.*

Proof. For the first statement look at Figure 5.2a. Here $p_3 \multimap p_4$ but $p_4 \not\multimap p_3$ since $p_4 \multimap p_5$. For the second statement, observe that, due to its definition, $\multimap$ is right-unique, i.e., there is at most one point on the right-hand side of the relation. Thus, $\multimap$ is a partial function. □

Using the $\multimap$ relation we now construct an undirected graph that relates two points if and only if one is the unique nearest neighbor of the other:

▷ nearest neighbor graph (NNG)

Definition 5.5. Let $P \subseteq \mathbb{R}^d$ be a d-dimensional point set. Then, its *nearest neighbor graph (NNG)*, denoted by $\mathcal{NG}(P)$, is an undirected graph $G = (P, E)$. We have an edge between two points if one of the two is the unique nearest neighbor of the other, i.e.,

$$E = \left\{ \{p, q\} \in \binom{V}{2} \;\middle|\; p \multimap q \vee q \multimap p \right\}.$$

Since the node set of the NNG is the same as the input, we generally only care about the edges of the NNG. We thus use the notation $\mathcal{N}(P)$ to denote the edges E of $\mathcal{NG}(P)$.

Even though the nearest neighbor graph is an undirected graph, we will often depict nearest neighbor graphs as directed graphs where there is an edge (p, q) if and only if $p \multimap q$. The reason is that this way we can always extract the information which point connects to which other point in which color. This would not be possible if we only showed the NNG's undirected edges. See Figure 5.2a for an example NNG of five points.

5.2.2 Colored Nearest Neighbor Graphs

As described in the introduction, our goal is to assign colors to each point. For this we will be given a number of colors we are allowed to use, and each point should have at least one color and has otherwise no restrictions on the colors assigned. This is formalized in

Definition 5.6. Let $P \subseteq \mathbb{R}^d$ be a d-dimensional point set and let $\hat{c} \in \mathbb{N}^+$ be a number of colors. We define $C_{\hat{c}} = \{1, \dots, \hat{c}\}$ as the *color set* with $\hat{c}$ colors. A *color assignment* $\sigma : P \to 2^{C_{\hat{c}}} \setminus \varnothing$ is a function which assigns a non-empty subset of colors to each point. We denote by $P_{c \in \sigma} = \{p \in P \mid c \in \sigma(p)\}$ all points which have $c \in C_{\hat{c}}$ among their assigned colors. If σ is clear from the context, we shorten the notation to P_c. We call $P_{c\in\sigma}$ or P_c the *c-colored points* of P.

▷ color set
▷ color assignment
▷ colored points

An example color assignment can be seen in Figure 5.2b where different points are assigned different numbers of colors. The nearest neighbor graph of the colored points is then obtained by combining the nearest neighbor graphs for each color.

Definition 5.7. Let $P \subseteq \mathbb{R}^d$ be a d-dimensional point set and let $\sigma : P \to 2^C \setminus \varnothing$ be a color assignment. Then, for every color $c \in C$, let $E_c = \mathcal{N}(P_c)$ be the edges of the nearest neighbor graph of the points that have color c. Let then $\mathbf{E} = \biguplus_{c\in C} E_c$ where $\uplus$ is the multiset sum and $\mathbf{E}$ is the resulting multiset. Then, the *colored nearest neighbor multigraph (CNNM)* $\mathcal{NG}(P,\sigma)$ is the multigraph $(P, \mathbf{E})$; any edge that appears in multiple nearest neighbor graphs of different colors appears multiple times in $\mathcal{NG}(P,\sigma)$ as well.

If no edge appears more than once in $\mathcal{NG}(P,\sigma)$, it is interpreted as an ordinary graph and called a *colored nearest neighbor graph (CNNG)*. This is the case if $E_c \cap E_{c'} = \varnothing$ for all distinct $c, c' \in C$, or in other words, if $\biguplus_{c\in C} E_c = \bigcup_{c\in C} E_c$.

Since the node set of the CNNM is equal to the input points, we generally only care about the edges of the CNNM. We use the notation $\mathcal{N}(P,\sigma)$ to denote the edges $\mathbf{E}$ of $\mathcal{NG}(P,\sigma)$.

▷ colored nearest neighbor multigraph (CNNM)
▷ colored nearest neighbor graph (CNNG)

The CNNG of the example color assignment can be seen in Figure 5.2c. We can see that it is a union of the three NNGs with edges $E_1 = \{\{p_1,p_2\},\{p_3,p_4\}\}$, $E_2 = \{\{p_2,p_3\}\}$, and $E_3 = \{\{p_2,p_4\},\{p_4,p_5\}\}$.

Our focus will be on the colored nearest neighbor graphs (as opposed to colored nearest neighbor multigraphs) which, as defined, ensure that every edge exists because of exactly one color. However, some intermediate results work for CNNMs and are thus stated as such.

Sometimes it will be useful to quickly state that $p \circ\!\!\rightarrow q$ with respect to the c-colored points P_c for some color $c \in C$. We do this by saying $p \overset{c}{\circ\!\!\rightarrow} q$. If we need to make clear which color assignment σ we talk about, we indicate this by writing $p \underset{\sigma}{\overset{c}{\circ\!\!\rightarrow}} q$.

As a first observation for CNNMs we can see that a color that is assigned to fewer than two points can be ignored:

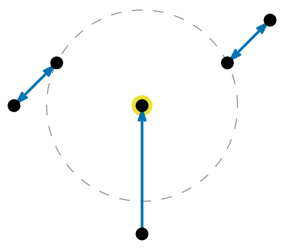


Figure 5.3: An example nearest neighbor graph of a two-dimensional point set where the highlighted point in the middle does not connect to any point since its nearest neighbor is not unique.

Observation 5.3. *Let $P \subseteq \mathbb{R}^d$ be a d-dimensional point set and let $\sigma : P \to 2^C \setminus \emptyset$ be a color assignment. If $|P_c| < 2$ for some $c \in C$ we can remove color c from σ to obtain $\sigma' : P \to 2^{C \setminus \{c\}} \setminus \emptyset$ with $\mathcal{N}(P, \sigma') = \mathcal{N}(P, \sigma)$.*

Proof. If $|P_c| < 2$ then $E_c = \emptyset$ as the NNG of an empty point set or a point set with just one point does not contain any edge. Thus, removing c from σ does not change the edges of the CNNM. □

As a result, for the rest of this work, we can assume that in any given color assignment each color is assigned to at least two points, unless specifically stated otherwise.

5.2.3 General Position

Let $P \subseteq \mathbb{R}^d$ be a d-dimensional point set and $p, q \in P$ two different points. The way we defined the $\multimap$ relation, we can see that $p \multimap q$ only if all other points are strictly further away from p. If there is another point $q' \in P \setminus \{p, q\}$ with $\text{dist}(p, q') = \text{dist}(p, q)$, then neither $p \multimap q$ nor $p \multimap q'$. This may lead to the situation that p does not connect to any other point (in some color that it is assigned). See Figure 5.3 for a two-dimensional example.

We want to ensure, however, that this situation does not arise in the situations that we analyze. Every point should always connect to exactly one point. That is, we only want to look at point sets P such that $\multimap$ is a total function with respect to P. Furthermore, it should also hold for any color assignment that $\overset{c}{\multimap}$ is a total function for all colors c.

The easiest way to ensure this is by guaranteeing that for every point the distances to all other points are all distinct. For point sets that do not fulfill this criterion, it can be achieved by a small random perturbation of each point.

▷ general position

Definition 5.8. A d-dimensional point set $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^d$ is in *general position* if and only if for all distinct $i, j, k \in [1..n]$ it holds that $\text{dist}(p_i, p_j) \neq \text{dist}(p_i, p_k)$.

5.3 One-Dimensional Colored Nearest Neighbor Graphs

We now look at specific properties of one-dimensional CNNGs. Here the given point set is a subset of $\mathbb{R}$. This has some direct implications that we will explore now. From now on we will assume that all point sets are in general position and also that they are already sorted from left to right. We define these two properties for our convenience as

▷ input point set

Definition 5.9. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be a one-dimensional point set. We call P an *input point set* if P is in general position and the elements in P are ordered, i.e., $p_1 < p_2 < \cdots < p_n$.

Then it is easy to obtain its nearest neighbor graph $\mathcal{N}(P)$:

Observation 5.4. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set. Every point has at most two candidates to which they can connect, those closest to it to the left and right. Thus, we can compute $\mathcal{N}(P)$ in $O(n)$ time.*

Proof. Since we have one-dimensional points we know that $p_1 \mathrel{\circ\!\!\longrightarrow} p_2$ and $p_n \mathrel{\circ\!\!\longrightarrow} p_{n-1}$. Furthermore, for all $i \in [2..(n-1)]$ we have either $p_i \mathrel{\circ\!\!\longrightarrow} p_{i-1}$ or $p_i \mathrel{\circ\!\!\longrightarrow} p_{i+1}$. For every point we can thus decide in constant time to which point it connects and add those edges to $\mathcal{N}(P)$, giving a linear time algorithm. □

As a result of this observation, we can see that the edge candidates come from a limited set of possible edges. Since every point can only connect to its immediate neighbors we can define this set as

▷ neighbor edges
▷ illegal edge

Definition 5.10. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set. We call the set

$$E_P = \{e_i = \{p_i, p_{i+1}\} \mid i \in [1..(n-1)]\}$$

the *neighbor edges* of P. All edges between points in P that are not in the set of neighbor edges are called *illegal* edges.

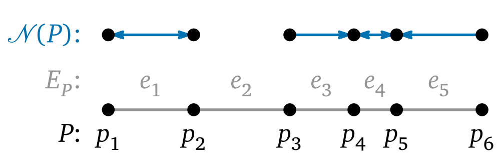
In Figure 5.4a we can see the neighbor edges of an example point set together with its NNG whose edges are a subset of E_P. Considering the additional colors we can now define the two problems that are the focus of the remaining chapters in this thesis.

5.3.1 Problem Statements

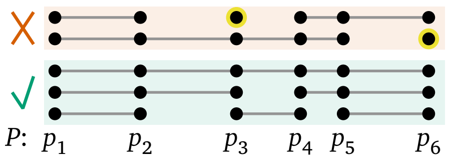
The first problem is about finding a color assignment such that the resulting graph contains exactly all neighbor edges.

Problem 5.1. Let $P \subseteq \mathbb{R}$ be an input point set and $\hat{c} \in \mathbb{N}^+$ be a number of colors. The 1-Dimensional Colored Nearest Neighbor Graph (1D-CNNG) problem is the

↪



(a) A one-dimensional point set P with all neighbor edges E_P. On top, we see $\mathcal{N}(P)$ which includes all but one edge of E_P.



(b) Different subsets $E \subseteq E_P$. The lower three are input edge sets, the upper two are not; the problematic points are highlighted.

Figure 5.4: A point set with its neighbor edges, its NNG, and some subsets of E_P.

problem of finding a color assignment σ with $\hat{c}$ colors such that $\mathcal{N}(P, \sigma) = E_P$, i.e., the resulting edges should be exactly the neighbor edges of P.

In addition, the 1-Dimensional Colored Nearest Neighbor Graph Optimization (1D-CNNG-Opt) problem is the problem of finding the smallest number $c^* \in \mathbb{N}^+$ such that there is a solution for the 1D-CNNG problem with $\hat{c} = c^*$.

In Figure 5.5a we depict an example input point set for the 1D-CNNG-Opt problem together with a possible solution. The figure shows the color assignment and the resulting CNNG. Since E_P is an ordinary set it is clear from the definition that any solution to Problem 5.1 must result in a CNNG and not just a CNNM.

For the second problem we want to generalize the 1D-CNNG and 1D-CNNG-Opt problem to subsets of the neighbor edges. Thus, we first define which subsets of the neighbor edges will be considered:

▷ input edge set

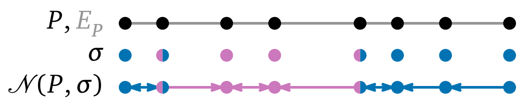
Definition 5.11. Let $P \subseteq \mathbb{R}$ be an input point set. Then a set $E \subseteq E_P$ is called an *input edge set* if and only if every point in P has at least one edge in E; more formally if and only if for all $p \in P$ there is an edge $\{q, r\} \in E$ such that $p = q$ or $p = r$.

When we write $e_i \in E$ or $e_i \notin E$, we mean that $e_i = \{p_i, p_{i+1}\}$, i.e., it is the ith edge in E_P from left to right.

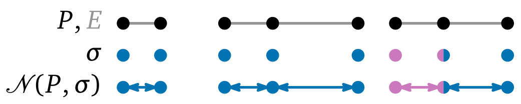
The idea here is that we allow the removal of edges; however, every point needs to have at least one incident edge: For every color assignment and every point p there is at least one color c assigned to the point and thus $p \overset{c}{\hookrightarrow} q$ for some other point q. See Figure 5.4b for example subsets of E_P, some of which are input edge sets and others which are not because there are isolated nodes.

Since every point in P has at least one incident edge, the union of all endpoints of the edges in E gives us the input point set P. It is thus generally not necessary to provide both P and E, and it suffices to provide only E. If, in the remainder of this thesis, we only say that we are given an input edge set $E \subseteq E_P$, it is clear that P is the underlying input point set.

We want to be able to reference those edges that are missing in an input set E. Therefore, we give those edges a special name:



(a) A two color solution for a 1D-CNNG-Opt instance.



(b) A two color solution for a 1D-CNNG-Gaps-Opt instance.

Figure 5.5: Two example instances for 1D-CNNG-Opt and 1D-CNNG-Gaps-Opt on the same point set. Both solutions are optimal and require two colors.

Definition 5.12. Let $E \subseteq E_P$ be an input edge set. Then any $e \in E_P \setminus E$ is called a *gap*. We define the set of *gaps* in E as $\mathcal{G}(E) = E_P \setminus E$, and we often use the slightly more concise notation $\mathcal{G}_E$. ▷ gaps

The second problem can now be defined as follows:

Problem 5.2. Let $E \subseteq E_P$ be an input edge set and $\hat{c} \in \mathbb{N}^+$ be a number of colors. The 1-Dimensional Colored Nearest Neighbor Graph With Gaps (1D-CNNG-Gaps) problem is the problem of finding a color assignment σ with $\hat{c}$ colors such that $\mathcal{N}(P,\sigma) = E$, i.e., the resulting edges should be exactly the input edge set.

In addition, the 1-Dimensional Colored Nearest Neighbor Graph With Gaps Optimization (1D-CNNG-Gaps-Opt) problem is the problem of finding the smallest number $c^* \in \mathbb{N}^+$ such that there is a solution for the 1D-CNNG-Gaps problem with $\hat{c} = c^*$.

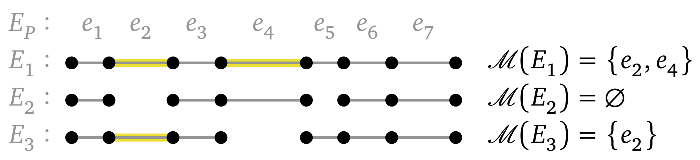
In Figure 5.5b we depict an example input point set with an input edge set as input to the 1D-CNNG-Opt problem. It also shows the color assignment and the resulting CNNG of a possible solution. Since E_P and thus also E is an ordinary set it is clear from the definition that any solution to Problem 5.2 must result in a CNNG and not just a CNNM.

To talk about the solutions and whether a color assignment is a solution for a specific problem instance, we define what it means for a color assignment to be valid:

Definition 5.13. Let $P \subseteq \mathbb{R}$ be an input point set. Any color assignment σ with $\mathcal{N}(P,\sigma) = E_P$ is called *valid for P*. If P and the fact that we talk about the 1D-CNNG problem is clear from context, we just say that σ is *valid*. ▷ valid

Let $E \subseteq E_P$ be an input edge set. Any color assignment σ with $\mathcal{N}(P,\sigma) = E$ is called *valid for E*. If E and the fact that we talk about the 1D-CNNG-Gaps problem is clear from context, we just say that σ is *valid*.

From the problem definitions it is obvious that 1D-CNNG-Gaps and 1D-CNNG-Gaps-Opt are at least as hard as 1D-CNNG and 1D-CNNG-Opt, respectively. Solving 1D-CNNG just means solving 1D-CNNG-Gaps for the input edge set $E = E_P$. The same holds true for the relation between 1D-CNNG-Gaps-Opt and 1D-CNNG-Opt.



E_P :
e_1 e_2 e_3 e_4 e_5 e_6 e_7
E_1 :
E_2 :
E_3 :
$\mathscr{M}(E_1) = \{e_2, e_4\}$
$\mathscr{M}(E_2) = \varnothing$
$\mathscr{M}(E_3) = \{e_2\}$

Figure 5.6: Three different input edge sets $E_i \subseteq E_P$ with their corresponding highlighted local maxima $\mathscr{M}(E_i)$.

As a result, the following chapters will mostly focus on 1D-CNNG-Gaps and 1D-CNNG-Gaps-Opt unless there is a distinction, as for example in Chapter 7.

5.3.2 Useful Definitions and Observations

Let $P = \{p_1, \dots, p_n\}$ be an input point set and E_P its neighbor edges. As the nearest neighbor graphs heavily depend on the distances between two points, we define $\|e_i\| = \operatorname{dist}(p_i, p_{i+1})$ for all edges $e_i = \{p_i, p_{i+1}\} \in E_P$ as the length of the edge.

For any input edge set we can check how long an edge is in relation to its adjacent edges, if they are present in the input edge set. If an edge is longer than both its adjacent edges (which is only possible if they are both present in the input edge set) we will generally treat it differently than the other edges. This warrants the following

▷ local maximum

Definition 5.14. Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$ with n points. An edge $e_i \in E$ is called a *local maximum* if and only if both neighbor edges are present in E and shorter than e_i. The set of all such edges is called the *local maxima* of E and referred to as $\mathscr{M}(E)$; we will often use the slightly more concise notation $\mathscr{M}_E$:

$$\mathscr{M}(E) = \left\{ e_i \in E \setminus \{e_1, e_{n-1}\} \;\middle|\; e_{i-1}, e_{i+1} \in E \wedge \|e_i\| > \max(\|e_{i-1}\|, \|e_{i+1}\|) \right\}.$$

Look at Figure 5.6 to see three different input edge sets and their respective highlighted local maxima. We can see that even though e_4 is a local maximum in E_1 and present in E_2 it is not a local maximum in E_2 since its adjacent edge e_5 is missing. Even though e_7 is longer than e_6 it is not a local maximum since it does not have an adjacent edge to its right. From $\mathscr{M}(E)$'s definition it follows immediately that we can quickly calculate it:

Observation 5.5. *Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$ with n points. Then $\mathscr{M}(E)$ can be computed in linear time $O(n)$.* □

So far we have defined a color assignment as a function which assigns a non-empty set of colors to each point. As we will now see, to solve both Problems 5.1 and 5.2 we can restrict this to assigning either one or two colors to each point:

Lemma 5.1. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set and $E \subseteq E_P$ be an input edge set. In any valid color assignment σ every point is assigned at most as many colors as it has incident edges in E.*

Proof. As stated after Observation 5.3 we can assume that σ assigns each of its colors to at least two points. Assume that σ assigns more colors to a point p_i than it has incident edges in E.

If p_i has two incident edges, assume without loss of generality that $\{1, 2, 3\} \subseteq \sigma(p_i)$. Since P is in general position it follows that there are $p_j, p_k, p_l \in P$ with $p_i \circ\!\xrightarrow{1} p_j$, $p_i \circ\!\xrightarrow{2} p_k$, and $p_i \circ\!\xrightarrow{3} p_l$. If two of those three points are equal it means that the resulting edge appears in two colors and thus the resulting CNNM is not a CNNG. If they are all distinct, only two of them can be p_{i-1} and p_{i+1}. Let without loss of generality p_j be the point such that $p_j \notin \{p_{i-1}, p_{i+1}\}$. Then the edge $\{p_i, p_j\}$ which is not in E will be present in $\mathcal{N}(P, \sigma)$. In both cases we can conclude that σ is not valid.

If p_i has only one incident edge, assume without loss of generality that $\{1, 2\} \subseteq \sigma(p_i)$. Then there are $p_j, p_k \in P$ with $p_i \circ\!\xrightarrow{1} p_j$ and $p_i \circ\!\xrightarrow{2} p_k$. As before, if $p_j = p_k$, the same edge appears in colors 1 and 2. On the other hand, if $p_j \neq p_k$ only one of them can be the endpoint of the only edge incident to p_i. Thus, the other point is not adjacent to p_i which means that an edge, that is not in E will be in $\mathcal{N}(P, \sigma)$. In both cases σ is not valid. $\square$

Even though Lemma 5.1 talks only about the 1D-CNNG-Gaps problem, it holds for the 1D-CNNG problem as well, since it is just a special case of the 1D-CNNG-Gaps problem. As a result of the lemma, given a color assignment $\sigma : P \to 2^{C_{\hat{c}}} \setminus \varnothing$ with $\hat{c}$ colors, we can constrain its codomain to all subsets of $C_{\hat{c}}$ of size one and two, denoted by $\mathbb{C}_{\hat{c}} = \{S \subseteq C_{\hat{c}} \mid 1 \leq |S| \leq 2\}$. Thus, σ can be seen as a function $\sigma : P \to \mathbb{C}_{\hat{c}}$ and in the remainder of this work we will treat every color assignment as such.

Chapter 6

Linear Time Algorithms for One and Two Colors

After having defined the problems that we want to solve, we first want to explore them slowly by starting with only few allowed colors. In Section 6.1 we will first see which inputs can be solved with just one color which is basically no color at all. This decision is easily done in linear time. Then, Section 6.2 explores how to color the endpoints of local maxima or edges that are not present in the input. This prepares us to introduce a restricted version of color assignments in Section 6.3 which turn out to be as powerful as normal color assignments. We finally solve the problems for two colors in Section 6.4 in linear time by constructing a helper graph that models the valid color assignments.

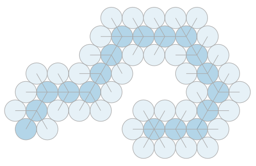
6.1 Solving One Color

We now start by looking at the case where we are only allowed to use one color to color our point set $P \subseteq \mathbb{R}$. This quickly turns out to be the same as constructing the NNG of the underlying point set, ignoring the color. Since a color assignment assigns at least one color to every point there is exactly one color assignment σ_1 with one color: the one which assigns this color 1 to every point. But then the point set P_1 for color 1 is exactly the same as P itself, meaning that $\mathcal{N}(P) = \mathcal{N}(P_1) = \mathcal{N}(P, \sigma_1)$. This is best summarized as

Observation 6.1. *Let $P \subseteq \mathbb{R}$ be an input point set. Then, for all color assignments σ_1 with one color, $\mathcal{N}(P) = \mathcal{N}(P, \sigma_1)$.* □

In our next step we look at what the NNG of a one-dimensional point set looks like. Since every point connects to the closer of its at most two neighbors it means that if two neighboring points are far apart that their connecting edge will not be present in the NNG. This means that none of the local maxima of the neighbor edges will be in $\mathcal{N}(P)$, as shown in

Lemma 6.1. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set. Then $\mathcal{N}(P) = E_P \setminus \mathcal{M}(E_P)$, that is, P's nearest neighbor graph contains all neighbor edges except the local maxima.*



Proof. Let $e_i \in E_P$ be a neighbor edge of P. We first look at the case that $e_i \in \mathcal{M}(E_P)$. This means that e_i is neither the leftmost nor the rightmost edge and that it is longer than both its adjacent edges. More formally, $1 < i < n-1$ and $\|e_i\| > \max(\|e_{i-1}\|, \|e_{i+1}\|)$. Since e_i is longer than its adjacent edges, we know that p_i is closer to p_{i-1} than to p_{i+1} which implies that $p_i \multimap p_{i-1}$. Similarly, p_{i+1} is closer to p_{i+2} than to p_i, meaning that $p_{i+1} \multimap p_{i+2}$. It follows that the edge $\{p_i, p_{i+1}\} = e_i$ does not appear in $\mathcal{N}(P)$. Have a look at Figure 6.2a to see an example of such an edge.

We now look at the case that $e_i \notin \mathcal{M}(E_P)$, which means that the negation of the above formal statement is true. If $i = 1$ or $\|e_i\| < \|e_{i-1}\|$ it is clear that $p_i \multimap p_{i+1}$. If $i = n-1$ or $\|e_i\| < \|e_{i+1}\|$ it follows that $p_{i+1} \multimap p_i$. In all cases we can conclude that $e_i \in \mathcal{N}(P)$.

Thus, for all $e_i \in E_P$ we have that $e_i \in \mathcal{N}(P)$ if and only if $e_i \notin \mathcal{M}(E_P)$ which concludes the proof. □

With this knowledge we can now easily compute the NNG of a point set. To decide whether we can obtain a given input edge set we just check whether the input edge set is the same as the NNG of the point set. This is summarized in

Lemma 6.2. *Let $P \subseteq \mathbb{R}$ be an input point set and $E \subseteq E_P$ be an input edge set. We can solve the* 1D-CNNG-Gaps *problem for E and $\hat{c} = 1$ in linear time.*

Proof. Observation 6.1 tells us that the CNNM for one color is exactly the NNG of the point sets. Thus, we can just look at $\mathcal{N}(P)$. Lemma 6.1 tells us that $\mathcal{N}(P) = E_P \setminus \mathcal{M}(E_P)$. From Observation 5.5 we know that we can compute $\mathcal{M}(E_P)$ in linear time with which we compute $\mathcal{N}(P)$ in linear time. We then only need to return whether $\mathcal{N}(P) = E$. □

6.2 Coloring Local Maxima and Gaps

As seen in the previous section, we cannot color local maxima with just one color such that they appear in the resulting CNNG. We will first make some observations about restrictions that arise from wanting a point to connect to a specific other point. This information will be used for observing in which ways local maxima and also gaps can or must be colored to obtain a valid color assignment. In the end we will generalize these restrictions so that the concept can be used later on in the algorithm.

6.2.1 Restrictions on Other Points

When constructing a color assignment, we will do it in a way by stating that a point p should connect to another point q in color c. This is, of course, only possible if all other points r with $\operatorname{dist}(p, r) < \operatorname{dist}(p, q)$ do not have color c. Since p should only ever connect to its left or right neighbor point, we can see that the condition is trivially met for all points on the same side of p as q: all points different from q are further

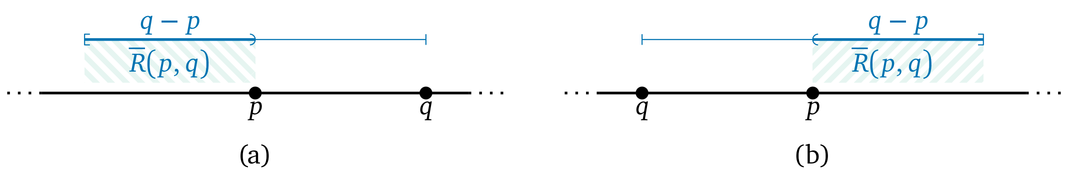


Figure 6.1: Two examples of a restricted region, (a) one on the left and (b) one on the right.

away from p. The points on the other side of p, however, may interfere. This is exactly the problem with local maxima: Let $e_i = \{p_i, p_{i+1}\}$ be a local maximum. Then p_i is always closer to p_{i-1} than to p_{i+1} and if p_i is supposed to connect to p_{i+1} in color c then p_{i-1} cannot have color c. In fact, there may be an arbitrary number k such that p_i is closer to all points $p_{i-k}, \dots, p_{i-1}$ than to p_{i+1}, and they must all be excluded from having color c assigned. This is formalized by the following definition for which two examples can be found in Figure 6.1.

Definition 6.1. Let the set of all half-open intervals on $\mathbb{R}$ be defined as

▷ restricted region

$$\mathbb{I} = \bigcup_{a,b \in \mathbb{R} \wedge a<b} (a,b] \cup [a,b).$$

We then define the *restricted region* of two points $p, q \in \mathbb{R}$ with $p \neq q$ as the function $\overline{R} : \mathbb{R} \times \mathbb{R} \to \mathbb{I}$. The result is the half-open set of points $r \in \mathbb{R}$ to which p is at least as close as to q and which are not on the same side of p as q. Or more formally:

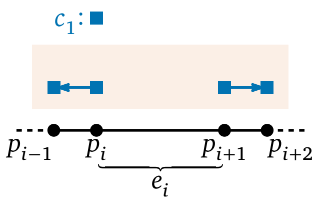
$$\overline{R}(p,q) = \begin{cases} [p - (q-p), p) = [2p - q, p) & \text{if } p < q, \\ (p, p + (p-q)] = (p, 2p - q] & \text{if } p > q. \end{cases}$$

With this definition we can now easily describe the constraints that arise from wanting a point p_i to connect to its neighbor p_{i+1} in color c: We must ensure that no point in $P \cap \overline{R}(p_i, p_{i+1})$ has color c among its colors.

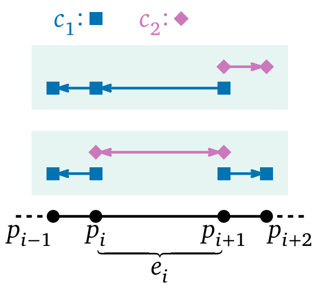
6.2.2 Coloring Local Maxima

We have seen in Lemma 6.1 that coloring with just one color will always exclude the local maxima of E_P. This can now easily be explained by looking at the restricted regions. Let $e_i = \{p_i, p_{i+1}\}$ be a local maximum. In order for $p_i \multimap p_{i+1}$ to happen, there can be no point inside $\overline{R}(p_i, p_{i+1})$. However, since e_i is a local maximum, it is always the case that $p_{i-1} \in \overline{R}(p_i, p_{i+1})$. Due to symmetry, the same holds when looking at p_{i+2}. Thus, to color the endpoints of a local maximum such that the local maximum is present in a CNNM we are forced to use more than one color.

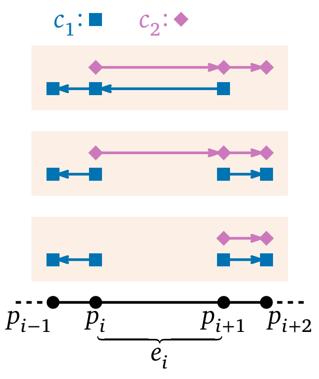
Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set and let $e_i \in E_P$ be a local maximum. First, we only consider all possible ways to color the endpoints of e_i and its two adjacent



(a) The use of only one color excludes edge e_i as seen in Lemma 6.1.



(b) These two situations result in a colored nearest neighbor graph.



(c) These results are either no CNNGs or do not include the edge e_i.

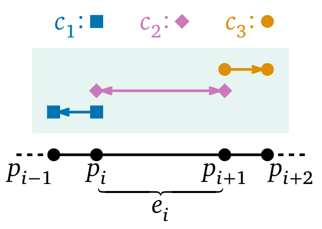
Figure 6.2: The different ways to color the endpoints of a local maximum e_i and the adjacent points with up to two colors. Each box shows one possible way to color. The colors above each point are those assigned to it. The arrows indicate which point connects to which other point in each color. We only show where p_i and p_{i+1} connect to.

points with up to two colors. In Figure 6.2 we can find a depiction of all structurally different possibilities without illegal edges. There we see the input point set and input edge set in black on the bottom of each figure. Above it, we see up to three boxes, each of which represents one possible (part of a) color assignment. For each color assignment we introduce one layer per color: if a point p_i is given this color, we place a colored marker on this layer above p_i. The arrows then represent the NNG for each individual color which makes it easy to see whether combining those NNGs into a CNNM results in the desired CNNG.

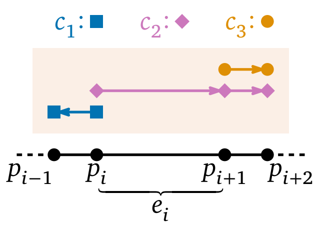
We notice that the figure only has arrows going out from p_i and p_{i+1}. The reason is that if p_{i-2}, p_{i-1}, and p_i have a common color c, then p_{i-1} connects to p_i or to p_{i-2} depending on which one is closer. Since it may be that $p_{i-1} \multimap p_{i-2}$, we cannot expect that e_{i-1} will be present due to p_{i-1} connecting to p_i. We can only be sure that e_{i-1} is present by enforcing that $p_i \multimap p_{i-1}$ is true. Using symmetric arguments, the same holds true for p_{i+1} and p_{i+2}. As a result the figure also ignores a color that may be given to p_{i-1} or p_{i+2} if it is not also a color for p_i or p_{i+1}, respectively.

Even though there are $3^4 = 81$ possible ways to color four points with two colors, we only show 6 in Figure 6.2. We have removed all structural duplicates, that is, the situations that can be obtained by mirroring the color assignment or swapping the two colors. We have also ignored the cases that have at least one the following properties:

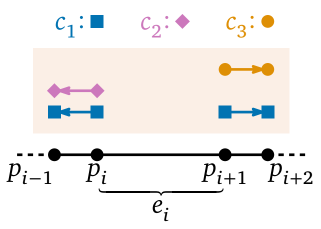
(1) *A color c is given to exactly one of the four points.* Call this point p_k. If no other point in P has this color, we can just remove the color and obtain the same CNNM, as seen in Observation 5.3. If p_k is either p_i or p_{i+1} the resulting edge is illegal



(a) This is the only solution with three colors that results in a CNNG.



(b) Edge e_{i+1} appears twice.



(c) Edge e_i is not present and both e_{i-1} and e_{i+1} appear twice.

Figure 6.3: The additional possible color assignments for a local maximum if we allow three or more colors.

as is goes to some p_j with $j < i - 1$ or $j > i + 2$. Otherwise, i.e., if p_k is either p_{i-1} or p_{i+2}, this is covered by property (3).

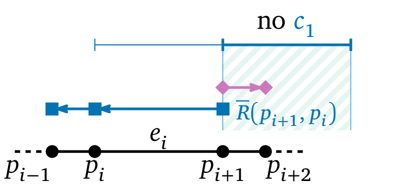
(2) *Two adjacent points do not have a common color.* This means that there will be no edge between these points.

(3) *One of the outer points, p_{i-1} or p_{i+2}, has a color that their adjacent inner point does not have.* The point will then either connect to the other inner or outer point, giving an illegal edge, or it will connect to the outside of the shown four points. As said before, this can then be ignored and the color removed.

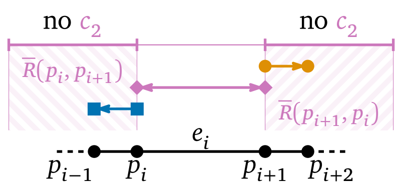
(4) *Each point is given both colors.* In this case, we have the edges shown in Figure 6.2a with the only difference that both edges appear in both colors.

The code used to enumerate all possible color assignments and then ignore those with certain properties can be found in Listing A.1. As we can see in Figure 6.2 there are then only two possible ways to color the points of and adjacent to a local maximum e_i such that the result is a CNNG, namely the two ways in Figure 6.2b. Even if we allow more than two colors, we can observe that more than three colors cannot happen: To not have property (2) the two endpoints of e_i need to share a common color, meaning that they can have at most three colors in total. In addition, property (3) forbids us that an outer point has a color their adjacent inner point does not have. Thus, only up to three colors are possible. Ignoring, as before, the situations which are obtained by mirroring or renaming the colors, there are only three additional color assignments with three colors that do not fulfill any of properties (1) to (3). These three can be seen in Figure 6.3 and the code to enumerate them is found in Listing A.1. Only the color assignment in Figure 6.3a yields the desired CNNG.

As a result, the three color assignments of a local maximum that yield a CNNG are those in Figures 6.2b and 6.3a. They can be divided into two different kinds of color assignments:



(a) If color assignment (i) is chosen, the choice of colors is restricted on one side of the local maximum.



(b) If color assignment (ii) is chosen, the choice of colors is restricted on both sides of the local maximum.

Figure 6.4: The constraints imposed by the two different color assignments for local maxima with two colors. If a point inside the hatched area had the forbidden color, the respective endpoint of the local maximum e_i would connect to it instead of the other endpoint of e_i.

(i) Either p_i or p_{i+1} has two colors and one of the colors is chosen for the points to its left and the other for the points to its right. This is the upper part of Figure 6.2b.

(ii) Both p_i and p_{i+1} have two colors such that at least one color c is shared by both (if both colors are shared, pick one as c). The color of p_i that is not c is used for p_{i-1} and the color of p_{i+1} that is not c is used for p_{i+2}. This is the situation in Figure 6.3a and the lower part of Figure 6.2b.

So far we have ignored all points to the left of p_{i-1} and to the right of p_{i+2}. However, as seen in Section 6.2.1 in order for an edge e_i to appear due to p_i connecting to p_{i+1} with color c it must be that no point inside $\overline{R}(p_i, p_{i+1})$ is allowed to be assigned color c. In Figure 6.4 we can see visualizations of these restricted regions. For each restricted region, the thin line segment above the local maximum indicates the two points whose distance is the length of the restricted region. The thicker line segment and the hatched area have the same length and indicate the actual restricted region, as labeled. The color also indicates the color that no point inside this region is allowed to have and this is also stated above the hatched region.

On the one hand Figure 6.4a shows the situation for color assignment (i) where only one endpoint has two colors and thus only $p_{i+1} \overset{c_1}{\circ\!\!\longrightarrow} p_i$ but not $p_i \overset{c_1}{\circ\!\!\longrightarrow} p_{i+1}$. As a result, there is only one restricted region $\overline{R}(p_{i+1}, p_i)$ to the right of the local maximum. In Figure 6.4b, on the other hand, we see the situation for color assignment (ii) where both endpoints have two colors and thus both $p_i \overset{c_2}{\circ\!\!\longrightarrow} p_{i+1}$ and $p_{i+1} \overset{c_2}{\circ\!\!\longrightarrow} p_i$. Consequently, there are restricted regions on both sides of the local maximum.

Since the constraints imposed by color assignment (ii) are a strict superset of the constraints imposed by color assignment (i), it seems to be never worse to choose color assignment (i) over color assignment (ii) for a local maximum. Indeed, as we will see in Section 6.3.1, we will be able to focus solely on color assignments that color local maxima with color assignment (i). However, we will first look at how to color the endpoints of gaps in the input edge set.

6.2.3 Coloring Gaps

As seen in Lemma 6.1, coloring with only one color will exclude all local maxima in E_P from the CNNG. As a result, if the input edge set E has a gap that is a local maximum in E_P it is easily excluded from the CNNG by giving both endpoints the same color. If the gap is not such a local maximum, we will need a change of color. To distinguish between these two types of gaps we give them special names:

▷ big gaps
▷ small gaps

Definition 6.2. Let $P \subseteq \mathbb{R}$ be an input point set and $E \subseteq E_P$ be an input edge set. A gap $e \in \mathcal{G}_E$ is called *big* if it is a local maximum in E_P, i.e., $e \in \mathcal{M}(E_P)$, and *small* otherwise. We denote the set of *small gaps* in E by $\mathcal{G}^-(E) = \mathcal{G}(E) \setminus E_P$, and we often use the slightly more concise notation $\mathcal{G}_E^-$.

Since we have already established that the endpoints of a gap both have exactly one color (see Lemma 5.1) the only question that remains is whether both of them should be given the same or different colors. With the following observation we can see which choice is possible for which kind of gap. In Figure 6.5 we can also see depictions of the four cases. In contrast to local maxima, where the task is to bring one endpoint to connect to the other endpoint, for gaps we need to prevent the endpoints from connecting to each other. To put it differently, for a gap $e_i = \{p_i, p_{i+1}\}$ we want that $p_i \overset{c}{\circ\!\!\rightarrow} p_{i-1}$ and $p_{i+1} \overset{c'}{\circ\!\!\rightarrow} p_{i+2}$ for (not necessarily distinct) colors $c, c' \in C_{\hat{c}}$. As a result, the restricted regions shown in Figure 6.5 are not to the left and right of the gap but start at one of the endpoints and go towards the other endpoint. It depends on the length of the edge next to the gap, whether the other endpoint is contained in the restricted region.

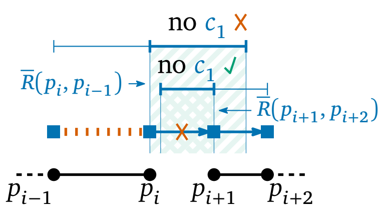
Observation 6.2. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a valid color assignment. Then for all gaps $e_i = \{p_i, p_{i+1}\} \in \mathcal{G}_E$ the following holds:*

1. *If e_i is small, it follows that $\sigma(p_i) \neq \sigma(p_{i+1})$.*
2. *If e_i is big, it can be that either $\sigma(p_i) \neq \sigma(p_{i+1})$ or $\sigma(p_i) = \sigma(p_{i+1})$.*

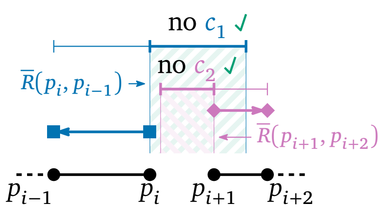
Proof. We first observe that $e_{i-1}, e_{i+1} \in E$ since E is an input edge set. From Lemma 5.1 we also know that $|\sigma(p_i)| = |\sigma(p_{i+1})| = 1$.

If e_i is small we know that $p_i \circ\!\!\longrightarrow p_{i+1}$ or $p_{i+1} \circ\!\!\longrightarrow p_i$. Thus, if it were that $\sigma(p_i) = \sigma(p_{i+1}) = \{c\}$ for some $c \in C$ it would follow that $p_i \overset{c}{\circ\!\!\rightarrow} p_{i+1}$ or $p_{i+1} \overset{c}{\circ\!\!\rightarrow} p_i$ and thus $e_i \in E_c \subseteq E$. See Figure 6.5a for an example where we can see that $p_i \overset{c_1}{\circ\!\!\rightarrow} p_{i+1}$ instead of the desired $p_i \overset{c_1}{\circ\!\!\rightarrow} p_{i-1}$. It must thus follow that $\sigma(p_i) \neq \sigma(p_{i+1})$, for which we can see in Figure 6.5b that then there is no problem.

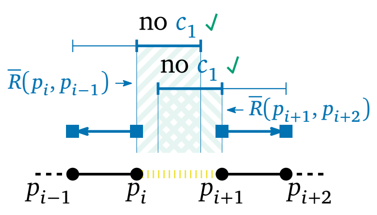
↪


no c_1 ✗
no c_1 ✓
$\overline{R}(p_i, p_{i-1})$
$\overline{R}(p_{i+1}, p_{i+2})$
p_{i-1}
p_i
p_{i+1}
p_{i+2}

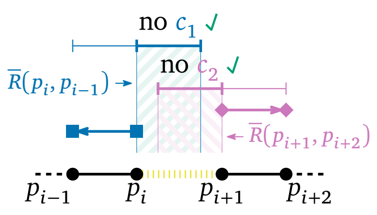
(a) The left edge is larger than the small gap. Giving the same color to p_i and p_{i+1} results in the wrong edge from p_i as it is closer to p_{i+1} than to p_{i-1}.


no c_1 ✓
no c_2 ✓
$\overline{R}(p_i, p_{i-1})$
$\overline{R}(p_{i+1}, p_{i+2})$
p_{i-1}
p_i
p_{i+1}
p_{i+2}

(b) If both endpoints of a small gap are given different colors, then p_i and p_{i+1} will not connect to each other, exactly as desired.


no c_1 ✓
no c_1 ✓
$\overline{R}(p_i, p_{i-1})$
$\overline{R}(p_{i+1}, p_{i+2})$
p_{i-1}
p_i
p_{i+1}
p_{i+2}

(c) A big gap can have the same color for both endpoints as they are further away from each other than their respective other neighboring points.


no c_1 ✓
no c_2 ✓
$\overline{R}(p_i, p_{i-1})$
$\overline{R}(p_{i+1}, p_{i+2})$
p_{i-1}
p_i
p_{i+1}
p_{i+2}

(d) Since the same color for both endpoints works for big gaps, as seen in (c), then giving different colors works as well.

Figure 6.5: The results of coloring the endpoints of a small gap, see (a) and (b), and a big gap, see (c) and (d), with the same or with different colors.

If, on the other hand, e_i is big we know that $p_i \circ\!\!\rightarrow p_{i-1}$ and $p_{i+1} \circ\!\!\rightarrow p_{i+2}$. If $\sigma(p_i) = \sigma(p_{i+1}) = \{c\}$ for some $c \in C$, it follows that $c \in \sigma(P_{i-1})$ and $c \in \sigma(P_{i+2})$ otherwise σ wouldn't be valid. It then implies that $p_i \circ\!\!\xrightarrow{c} p_{i-1}$ and $p_{i+1} \circ\!\!\xrightarrow{c} p_{i+2}$, as can be seen in Figure 6.5c. On the other hand, if $\sigma(p_i) = \{c\} \neq \{c'\} = \sigma(p_{i+1})$ it follows with the same arguments that $c \in \sigma(P_{i-1})$ and $c' \in \sigma(P_{i+2})$ and thus $p_i \circ\!\!\xrightarrow{c} p_{i-1}$ and $p_{i+1} \circ\!\!\xrightarrow{c'} p_{i+2}$. Figure 6.5d shows this situation. □

The problem in Figure 6.5a is that one of the restricted regions contains the other endpoint which has the same color. If we have a small gap this is always the case: if we both restricted regions do not contain the other endpoint then the gap is longer than both adjacent edges which makes it a big gap. Even though Figure 6.5 makes it look like we only need to check whether p_i and p_{i+1} lie inside the restricted regions associated with the other endpoint, this is not the case. There may be more points inside these regions, especially if for example e_{i-1} is very large compared to e_i. If one of those points has the same color as p_i it will also be that p_i connects to this point instead of p_{i-1}.

6.2.4 Restricted Regions for Input Edges and Gaps

We have seen in Sections 6.2.2 and 6.2.3 that both for local maxima and for gaps it is important to look at the restricted regions to determine whether a color assignment will result in the desired edges in the corresponding CNNM. However, the restricted regions differ between those of local maxima and those of gaps. For a local maximum (or more generally an edge) the restricted regions are defined by both endpoints while for gaps the restricted regions are defined by one endpoint and its adjacent point that is not the other endpoint. As a result, we want to homogenize the differences into combined functions.

▷ left restricted region
▷ right restricted region

Definition 6.3. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\mathbb{I}$ the half-open intervals on $\mathbb{R}$ as defined in Definition 6.1. We define the *left restricted region* $\overleftarrow{R}_E : E_P \to \mathbb{I}$ and the *right restricted region* $\overrightarrow{R}_E : E_P \to \mathbb{I}$ with respect to E as follows:

$$\overleftarrow{R}_E(e_i = \{p_i, p_{i+1}\}) = \begin{cases} \overline{R}(p_i, p_{i+1}) & \text{if } e_i \in E, \\ \overline{R}(p_{i+1}, p_{i+2}) & \text{if } e_i \notin E, \end{cases}$$

$$\overrightarrow{R}_E(e_i = \{p_i, p_{i+1}\}) = \begin{cases} \overline{R}(p_{i+1}, p_i) & \text{if } e_i \in E, \\ \overline{R}(p_i, p_{i-1}) & \text{if } e_i \notin E. \end{cases}$$

Even tough the left restricted regions start at different endpoints for edges and gaps they have in common that they extend towards the left, starting at one of the endpoints. Additionally, a point with the wrong color in a left restricted region will prevent left starting endpoint from connecting in this color to the point to its right. The mirrored observation holds true for the right restricted regions.

6.3 Simplifying the Color Assignments

Given a point set there are a lot of possible color assignments. From all of those we are only interested in the color assignments where two consecutive points are assigned a common color if the edge between them is in the input edge set. An upper bound on the number of possible color assignments is $\hat{c}^n$: Think about coloring from left to right. Then a point with both incident edges in the input edge set can have either a single color shared with the point to its left or an additional color from the remaining $\hat{c} - 1$ colors. A point that has only one incident edge either has its color fixed or can choose from $\hat{c} - 1$ different colors, resulting in at most $\hat{c}^n$ color assignments. This number is quite big and the color assignment have little structure. In this section we will define basic color assignments (BCAs) which will have as few color changes and points with more than one color as possible. This will also give them more structure we can then use for our linear time algorithm in Section 6.4. We will first observe in Section 6.3.1 that the only points that have to have two colors are endpoints of local maxima (and only one of the two endpoints). We will show that any valid color assignment can be transformed into a valid color assignment that fulfills this criterion. In Section 6.3.2 we will then see that only small gaps need to have different colors for their endpoints. This is also done by transforming valid color assignments that don't fulfill this criterion into ones that do. With this we will then be able to define BCAs in Section 6.3.3. This will allow us to focus only on those specific color assignments, not only in the upcoming algorithm but also in the remainder of this thesis.

6.3.1 Reducing the Number of Points With Two Colors

Our first goal is to transform any given color assignment into a color assignment that has the lowest possible number of points with two colors. To do this, we will take a color assignment and transform it into a new color assignment which reduces the number of colors from two to one for exactly one point. For all other points, the number of colors will remain the same.

▷ *i*-reduction
▷ $\text{swap}_{\ell,r}$

Definition 6.4. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set and $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a color assignment with $\hat{c} \geq 2$ colors. Let $p_i \in P$ be a point with two colors $\sigma(p_i) = \{\ell, r\}$, such that $p_i \overset{\ell}{\circ\!\!\rightarrow} p_{i-1}$ and $p_i \overset{r}{\circ\!\!\rightarrow} p_{i+1}$. We then define the color assignment $\sigma_i : P \to \mathbb{C}_{\hat{c}}$ as follows and call it the *i-reduction* of σ:

$$\sigma_i(p_j) = \begin{cases} \sigma(p_j) & \text{if } j < i, \\ \{\ell\} & \text{if } j = i, \\ \{\text{swap}_{\ell,r}(c) \mid c \in \sigma(p_j)\} & \text{if } j > i, \end{cases}$$

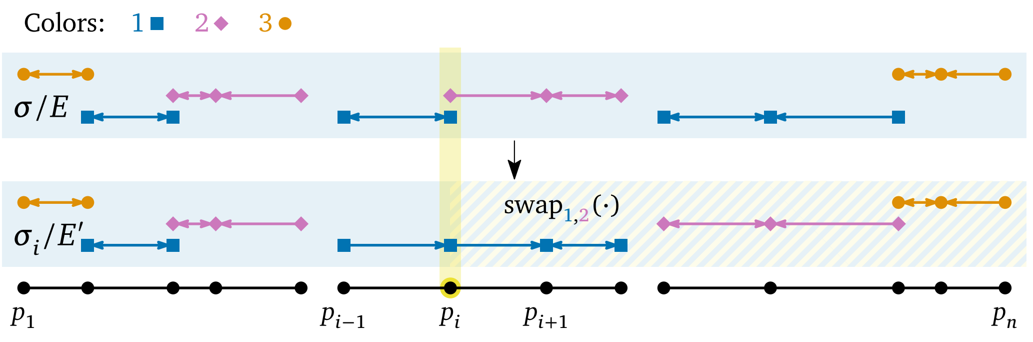
$$\text{with } \text{swap}_{\ell,r}(c) = \begin{cases} \ell & \text{if } c = r, \\ r & \text{if } c = \ell, \\ c & \text{otherwise.} \end{cases}$$

In words: The *i*-reduction of a color assignment σ removes the color from p_i with which it connects to the right. Furthermore, for all points to the right of p_i the two colors from p_i in σ are swapped. See Figure 6.6 for a color assignment and three example *i*-reductions for different points p_i. The main advantage of Definition 6.4 is that σ_i has exactly one point less with two colors than σ. However, to be able to use this, we need to show that σ_i results in the same CNNG as σ, at least for those σ that are valid. We will show that this is the case for most choices of p_i:

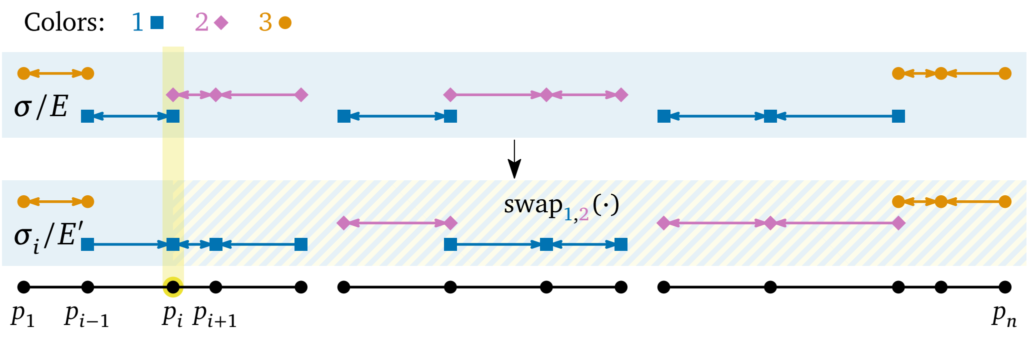
Lemma 6.3. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a valid color assignment. Let p_i be a point with two colors in σ and σ_i the i-reduction of σ. Then $\mathcal{N}(P, \sigma_i) = \mathcal{N}(P, \sigma)$, which implies that σ_i is valid, if and only if*

(a) p_i is not the endpoint of a local maximum in E, or

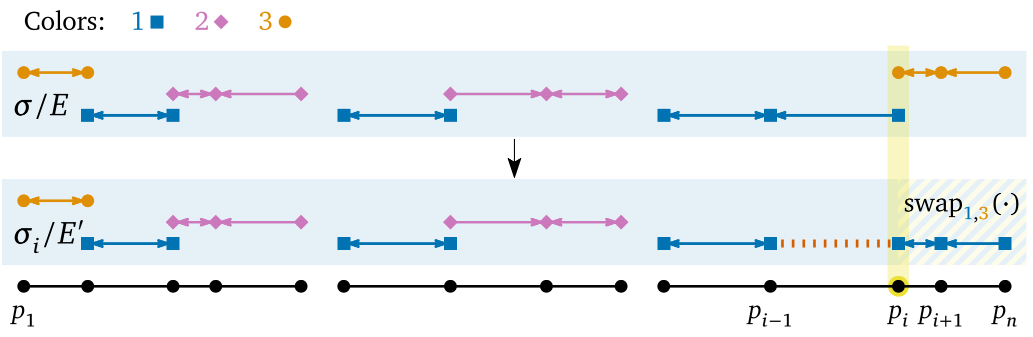
↪



(a) If p_i is not the endpoint of a local maximum we obtain the same edges for both color assignments σ and σ_i.



(b) Here p_i is the endpoint of a local maximum and the other endpoint p_{i-1} also has two colors. Again, we obtain the same edges for σ and σ_i.



(c) If p_i is the endpoint of a local maximum and the other endpoint p_{i-1} does not have two colors, then the local maximum is missing in E' for σ_i.

Figure 6.6: A color assignment σ with its i-reduction σ_i for three different points p_i. In (a) and (b) p_i has colors 1 and 2, σ_i removes 2 from p_i, and swaps 1 and 2 to the right of p_i. In both examples we can see that σ_i produces the same CNNG as σ. For (c) we remove 3 from p_i and swap 1 and 3 to its right. Here, the resulting edges E' do not contain the local maximum $\{p_{i-1}, p_i\}$ because neither p_{i-1} nor p_i has two colors in σ_i.

(b) p_i is such an endpoint and the other endpoint also has two colors.

Proof. For the proof, let $E' = \mathcal{N}(P, \sigma_i)$, that is, we want to show that $E' = E$ if and only if assumption (a) or (b) holds.

$E' \neq E$ is implied if neither assumption (a) nor (b) is fulfilled. We first show that if p_i fulfills neither assumption (a) nor (b) then $E' \neq E$. If neither assumption (a) nor (b) are true for p_i it excludes all points that are not endpoints of local maxima and all endpoints of local maxima whose other endpoint also has two colors. As a result p_i must be an endpoint of a local maximum and its other endpoint must have only one color. The local maximum can be either to the left or to the right of p_i. Assume without loss of generality that it is e_i, that is, it is to the right of p_i. Then, the other endpoint of e_i is p_{i+1} and by assumption it has only one color. Thus, $|\sigma(p_{i+1})| = |\sigma_i(p_{i+1})| = 1$. Additionally, $|\sigma_i(p_i)| = 1$ by Definition 6.4. This means that we have two endpoints of a local maximum, both with just one color.

If p_i and p_{i+1} have different colors, i.e., $\sigma_i(p_i) \neq \sigma_i(p_{i+1})$, then $e_i \notin E'$ and it follows that $E' \neq E$. Otherwise, let c be the common color, i.e., $\sigma_i(p_i) = \sigma_i(p_{i+1}) = \{c\}$. If $c \notin \sigma_i(p_{i-1})$ or $c \notin \sigma_i(p_{i+2})$ then $e_{i-1} \notin E'$ or $e_{i+1} \notin E'$, respectively, which also shows that $E' \neq E$. However, if $c \in \sigma_i(p_{i-1})$ and $c \in \sigma_i(p_{i+2})$ then $e_i \notin E'$ since $p_i \overset{c}{\underset{\sigma_i}{\circ\!\!\rightarrow}} p_{i-1}$ and $p_{i+1} \overset{c}{\underset{\sigma_i}{\circ\!\!\rightarrow}} p_{i+2}$. See Figure 6.6c for an example where $e_i \notin E'$.

$E' = E$ is implied if assumption (a) or (b) is fulfilled. We now show that if p_i fulfills assumption (a) or (b) then $E' = E$. An example for each assumption is shown in Figures 6.6a and 6.6b. Let $P_1 = P_{1\in\sigma}$ and $P_2 = P_{2\in\sigma}$ as well as $P'_1 = P_{1\in\sigma_i}$ and $P'_2 = P_{2\in\sigma_i}$ be the points with color 1 and 2 in σ and σ_i, respectively. Assume that, without loss of generality, $\sigma(p_i) = \{1, 2\}$, $p_i \overset{1}{\underset{\sigma}{\circ\!\!\rightarrow}} p_{i-1}$, and $p_i \overset{2}{\underset{\sigma}{\circ\!\!\rightarrow}} p_{i+1}$. Since σ_i only changes colors 1 and 2 it is obvious that for all $p \in P$ we have that $\sigma_i(p) \setminus \{1, 2\} = \sigma(p) \setminus \{1, 2\}$. It follows that $E \setminus (E_1 \cup E_2) = E' \setminus (E'_1 \cup E'_2)$. In order for $E' = E$ to hold me must show that $E'_1 \cup E'_2 = E_1 \cup E_2$ and to show that $\mathcal{N}(P, \sigma_i)$ is a CNNG we additionally need to show that $E'_1 \cap E'_2 = \emptyset$. To do this we look at each point p_j and see how it changes (if at all) to which other points it connects with colors 1 and 2. In the following we will often have $c \in \{1, 2\}$ be one of the two interesting colors. We then use $\bar{c}$ to denote $\bar{c} = \mathrm{swap}_{1,2}(c)$, that is, the color from 1 and 2 that is not c.

For the following points $p_j \in P$ assume now that $1 \in \sigma(p_j)$ or $2 \in \sigma(p_j)$ as otherwise p_j cannot be the endpoint of an edge in E_1, E_2, E'_1 or E'_2. If $\sigma(p_j) \cap \{1, 2\} = \emptyset$ then it follows that $\sigma_i(p_j) \cap \{1, 2\} = \emptyset$ and thus it will connect to the same point(s) in its assigned color(s) for both σ and σ_i.

1. $j = 1$: We know that $\sigma(p_1) = \{c\}$ and also that $p_1 \overset{c}{\underset{\sigma}{\circ\!\!\rightarrow}} p_2$ and thus $c \in \sigma(p_2)$. If $i > 2$ both p_1 and p_2 keep their colors in σ_i and thus $p_1 \overset{c}{\underset{\sigma_i}{\circ\!\!\rightarrow}} p_2$. If $i = 2$ we know that $p_2 \overset{1}{\underset{\sigma}{\circ\!\!\rightarrow}} p_1$ by assumption and thus $c = 1$. Since $\sigma_i(p_i) = \{1\}$, it follows that ↪

also $p_1 \circ\!\!\xrightarrow[\sigma_i]{1} p_2$. Thus, $e_1 \in E_c$ implies that $e_1 \in E'_c$ and since p_1 has only one color it cannot be that $e_1 \in E'_{\bar{c}}$.

2. $1 < j < i-1$: We know that σ_i is the same as σ for p_j and both its neighbors. For $c \in \{1,2\}$, we know that $p_j \circ\!\!\xrightarrow[\sigma]{c} p_{j-1}$ or $p_j \circ\!\!\xrightarrow[\sigma]{c} p_{j+1}$ if $c \in \sigma(p_j)$.

   If $p_j \circ\!\!\xrightarrow[\sigma]{c} p_{j+1}$ then we know that $\overleftarrow{R}_E(e_j) \cap P_c = \emptyset$. Since all points in $\overleftarrow{R}_E(e_j)$ have the same colors in σ and σ_i it follows that $\overleftarrow{R}_E(e_j) \cap P'_c = \emptyset$ as well. This means that $p_j \circ\!\!\xrightarrow[\sigma_i]{c} p_{j+1}$.

   If $p_j \circ\!\!\xrightarrow[\sigma]{c} p_{j-1}$ then we know that $\overrightarrow{R}_E(e_j) \cap P_c = \emptyset$. It thus cannot be that $p_i \in \overrightarrow{R}_E(e_j)$ since $\sigma(p_i) = \{1,2\}$. As a result, all points in $\overrightarrow{R}_E(e_j)$ have the same colors in σ and σ_i it follows that $\overrightarrow{R}_E(e_j) \cap P'_c = \emptyset$ as well. This means that $p_j \circ\!\!\xrightarrow[\sigma_i]{c} p_{j-1}$.

3. $j = i-1$: We know that $1 \in \sigma(p_{i-1})$ by assumption which implies $1 \in \sigma_i(p_{i-1})$. We also know that $1 \in \sigma(p_i)$ and $1 \in \sigma_i(p_i)$. Furthermore, σ_i and σ are the same for all points to the left, especially p_{i-2}. Then, $p_{i-1} \circ\!\!\xrightarrow[\sigma]{1} p_i$ directly implies $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{1} p_i$ and $p_{i-1} \circ\!\!\xrightarrow[\sigma]{1} p_{i-2}$ directly implies $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{1} p_{i-2}$.

   If $2 \in \sigma(p_{i-1})$ it must be that $p_{i-1} \circ\!\!\xrightarrow[\sigma]{2} p_{i-2}$ as otherwise e_{i-1} would appear twice in E. Since $2 \in \sigma(p_i)$ as well, it must then also be that p_{i-2} is closer to p_{i-1} than p_i. Then we have that $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{2} p_{i-2}$.

4. $j = i$: For p_i we know that $p_i \circ\!\!\xrightarrow[\sigma]{1} p_{i-1}$ and $p_i \circ\!\!\xrightarrow[\sigma]{2} p_{i+1}$. Since $\sigma_i(p_i) = \{1\}$, we know that p_i can now only connect to one instead of both neighbor points. We know, however, that $1 \in \sigma_i(p_{i-1})$ and $1 \in \sigma_i(p_{i+1})$ due to the definition of σ_i. Thus, p_i cannot be the endpoint of an illegal edge in E'_1. We will now show that $p_i \circ\!\!\xrightarrow[\sigma_i]{1} p_{i-1}$ implies $p_{i+1} \circ\!\!\xrightarrow[\sigma_i]{1} p_i$ and that $p_i \circ\!\!\xrightarrow[\sigma_i]{1} p_{i+1}$ implies $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{1} p_i$. Then we will know that both edges e_{i-1} and e_i that are present in E are also present in E'.

   If $p_i \circ\!\!\xrightarrow[\sigma_i]{1} p_{i+1}$, it means that p_i is closer to p_{i+1} than to p_{i-1}. See Figure 6.6 for both of the following situations. In case assumption (a) holds, i.e., p_i is not the endpoint of a local maximum, there are two possible explanations: One possibility is that p_{i-1} is closer to p_i than to p_{i-2} which then automatically implies that $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{1} p_i$. The other possibility is that the edge e_{i-2} to the left of p_{i-1} is not present in E, that is, it is a gap. This means that $p_{i-1} \circ\!\!\xrightarrow[\sigma]{1} p_i$ and no point in $\overline{R}(p_{i-1}, p_i)$, which is to the left of p_{i-1} has color 1 in σ. But then the same holds for σ_i since it assigns the same colors as σ for all points to the left of p_i. Thus, it also follows that $p_{i-1} \circ\!\!\xrightarrow[\sigma_i]{1} p_i$.

   If, on the other hand, assumption (b) holds, it must be that e_{i-1} is the local maximum since $\|e_i\| < \|e_{i-1}\|$ because $p_i \circ\!\!\xrightarrow[\sigma_i]{1} p_{i+1}$. Then we know that $\sigma(p_{i-1}) = \sigma_i(p_{i-1}) = \{1, c\}$ for some color $c \neq 1$. It then follows that $p_{i-1} \circ\!\!\xrightarrow[\sigma]{c} p_{i-2}$ and

↪

$p_{i-1} \circ\!\xrightarrow[\sigma]{1} p_i$. The latter then also holds for σ_i since to the left of p_{i-1} all points have the same colors in σ and σ_i. We thus have $p_{i-1} \circ\!\xrightarrow[\sigma_i]{1} p_i$.

If $p_i \circ\!\xrightarrow[\sigma_i]{1} p_{i-1}$, it means that p_i is closer to p_{i-1} than to p_{i+1}. Here we have the symmetric situation as the one before with the only difference that to the right of p_i all points change colors 1 and 2 from σ to σ_i. This means, however, that relative to the other points to the right of p_i there is no change. Since we have not used any argument about the points to the right of p_i for the previous case we can use all the symmetric arguments for this case. This then shows that we will always have $p_{i+1} \circ\!\xrightarrow[\sigma_i]{1} p_i$.

5. $j = i + 1$: We know that $2 \in \sigma(p_{i+1})$ by assumption which implies that $1 \in \sigma_i(p_{i+1})$. We also know that $2 \in \sigma(p_i)$ and $1 \in \sigma_i(p_i)$. Additionally, if $2 \in \sigma(p_{i+2})$ we know that $1 \in \sigma_i(p_{i+2})$. Then, $p_{i+1} \circ\!\xrightarrow[\sigma]{2} p_i$ directly implies $p_{i+1} \circ\!\xrightarrow[\sigma_i]{1} p_i$ and $p_{i+1} \circ\!\xrightarrow[\sigma]{2} p_{i+2}$ directly implies $p_{i+1} \circ\!\xrightarrow[\sigma_i]{1} p_{i+2}$.

   If $1 \in \sigma(p_{i+1})$ it must be that $p_{i+1} \circ\!\xrightarrow[\sigma]{1} p_{i+2}$ as otherwise e_i would appear twice in E, in colors 1 and 2. Since $1 \in \sigma(p_i)$ as well, it must be that p_{i+1} is closer to p_{i+2} than to p_i. As a result we know that $2 \in \sigma_i(p_{i+1})$ and $2 \in \sigma_i(p_{i+2})$ and there is no point in $\overline{R}(p_{i+1}, p_{i+2})$. Then we have that $p_{i+1} \circ\!\xrightarrow[\sigma_i]{2} p_{i+2}$.

6. $i + 1 < j < n - 1$: We know that σ_i has colors 1 and 2 swapped for p_j and both its neighbors. For $c \in \{1, 2\}$, we know that $p_j \circ\!\xrightarrow[\sigma]{c} p_{j-1}$ or $p_j \circ\!\xrightarrow[\sigma]{c} p_{j+1}$ if $c \in \sigma(p_j)$.

   If $p_j \circ\!\xrightarrow[\sigma]{c} p_{j-1}$, then we know that $\overrightarrow{R}_E(e_{j-1}) \cap P_c = \varnothing$. Since to the right of p_i all colors 1 and 2 are swapped we know that $\overrightarrow{R}_E(e_{j-1}) \cap P'_{\bar{c}} = \varnothing$. It is also the case that $\bar{c} \in \sigma_i(p_j)$ and $\bar{c} \in \sigma_i(p_{j-1})$ which then means that $p_j \circ\!\xrightarrow[\sigma_i]{\bar{c}} p_{j-1}$.

   If $p_j \circ\!\xrightarrow[\sigma]{c} p_{j+1}$, then we know that $\overleftarrow{R}_E(e_{j-1}) \cap P_c = \varnothing$. It thus cannot be that $p_i \in \overleftarrow{R}_E(e_{j-1})$ since $\sigma(p_i) = \{1, 2\}$. As a result, all points in $\overleftarrow{R}_E(e_{j-1})$ have colors 1 and 2 swapped and thus $\overleftarrow{R}_E(e_{j-1}) \cap P'_{\bar{c}} = \varnothing$. It is also the case that $\bar{c} \in \sigma_i(p_j)$ and $\bar{c} \in \sigma_i(p_{j+1})$ which then means that $p_j \circ\!\xrightarrow[\sigma_i]{\bar{c}} p_{j+1}$.

7. $j = n$: We know that $\sigma(p_n) = \{c\}$ and also that $p_n \circ\!\xrightarrow[\sigma]{c} p_{n-1}$ and thus $c \in \sigma(p_{n-1})$. If $i < n-1$ both p_n and p_{n-1} have the same colors in σ_i but swapped and thus $p_n \circ\!\xrightarrow[\sigma_i]{\bar{c}} p_{n-1}$. If $i = n - 1$ we know that $p_{n-1} \circ\!\xrightarrow[\sigma]{2} p_n$ by assumption and thus $c = 2$. Since $\sigma_i(p_i) = \{1\}$, it follows that $p_n \circ\!\xrightarrow[\sigma_i]{1} p_{n-1}$ since $\sigma_i(p_n) = \{\bar{c}\} = \{1\}$. Thus, $e_{n-1} \in E_c$ implies that $e_{n-1} \in E'_{\bar{c}}$ and since p_n has only one color it cannot be that $e_{n-1} \in E'_c$.

For each p_j except p_i we have now shown that $p_j \circ\!\xrightarrow[\sigma]{c} p_k$ implies that $p_j \circ\!\xrightarrow[\sigma_i]{c'} p_k$ where $c' = c$ for $j < i$ and $c' = \bar{c}$ for $j > i$. This shows that those edges created by some $p_j \neq p_i$ in E are exactly those created by the same p_j in E'. It especially also shows that no edge $e_j \notin \{e_{i-1}, e_i\}$ appears twice in E'. For p_i we have shown that both its

↪

incident edges e_{i-1} and e_i are always present in E'. And since p_i has only one color it automatically follows that those edges only appear once in E'. Thus, we know that $E' = E$, as claimed. □

At the end of Section 6.2.2 we already said that we will be able to focus on color assignments where for each local maximum only one endpoint has two colors. Given a valid color assignment, we can exhaustively apply Lemma 6.3 which turns a point with two colors into a point with one color in each application. In the resulting color assignment, which is still valid, the only points left with two colors are endpoints of local maxima whose other endpoint has only one color. This means that in the resulting color assignment all local maxima are colored with color assignment (i) (see page 98). It has the advantage that there is only one restricted region for each local maximum instead of two, thus increasing the number of possible valid color assignments. We now have a tool that already simplifies the color assignments a lot, it remains to look at what happens at gaps.

6.3.2 Few Color Changes at Gaps

Observation 6.2 tells us that a small gap will always need two different colors for its endpoints but that a big gap may have the same or different colors for its endpoints. We will now see that for every valid color assignment with two different colors for the endpoints of a big gap, there exists a valid color assignment that assigns the same color to those endpoints.

Lemma 6.4. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a valid color assignment. If there is a big gap $e_i = \{p_i, p_{i+1}\} \in \mathcal{G}_E \setminus \mathcal{G}_E^-$ such that $\sigma(p_i) \neq \sigma(p_{i+1})$ then there exists a valid color assignment $\sigma' : P \to \mathbb{C}_{\hat{c}}$ with $\sigma'(p_i) = \sigma'(p_{i+1})$ and for all $p \in P$ we have that $|\sigma(p)| = |\sigma'(p)|$.*

Proof. Assume, without loss of generality, that $\sigma(p_i) = \{1\}$ and $\sigma(p_{i+1}) = \{2\}$. We then define σ' as follows (reusing $\mathrm{swap}_{1,2}(\cdot)$ from Definition 6.4):

$$\sigma'(p_j) = \begin{cases} \sigma(p_j) & \text{if } j \leq i, \\ \{\mathrm{swap}_{1,2}(c) \mid c \in \sigma(p_j)\} & \text{if } j > i. \end{cases}$$

It is then obvious that $|\sigma(p)| = |\sigma'(p)|$ for all $p \in P$ and also that $\sigma'(p_i) = \sigma'(p_{i+1})$. Let $\mathcal{N}(P, \sigma') = E'$. We then need to show that $E' = E$ which, since only colors 1 and 2 are different between σ and σ', means that we need to show that $E'_1 \cup E'_2 = E_1 \cup E_2$ and that $E'_1 \cap E'_2 = \emptyset$.

Edge e appears twice. First, assume that there is an edge $e = \{p_a, p_b\} \in E'_1 \cap E'_2$ with $a < b$. Then it must be that $\sigma'(p_a) = \sigma'(p_b) = \{1, 2\}$ which can only be true if $\sigma(p_a) = \sigma(p_b) = \{1, 2\}$ as well. Furthermore, there can be no point p_c between p_a

and p_b with $\{1,2\} \cap \sigma'(p_c) \neq \varnothing$ as then p_a and p_b would rather connect to p_c instead of each other in the color shared with p_c. The same then holds true for σ by definition of σ'.

For both $c = 1$ and $c = 2$ it must also be that $\overline{R}(p_a, p_b)$ or $\overline{R}(p_b, p_a)$ contains no point with color c: If both restricted regions contained a point with color c then p_a would connect in color c to this point in $\overline{R}(p_a, p_b)$ and p_b would connect to the one in $\overline{R}(p_b, p_a)$. Thus, e would not be present in color c, contradicting our assumption. Then, in σ we have that at least one of the two restricted regions contains no point in at least one of the two colors 1 and 2: If both restricted regions contained points with both colors in σ we can see the color swap in σ' can only make changes such all points with one color in one restricted region are changed to the other color. The other restricted region would still contain points of both colors, which would not allow for e to appear in both colors. As a result, there is one endpoint of e and one color from 1 and 2 such that the endpoint connects to the other endpoint of e in this color in $\mathcal{N}(P, \sigma)$. Then $e \in E$ which also means that $b = a + 1$.

If now e were not a local maximum then either one of the adjacent edges wouldn't be in E or one of e's endpoints would be closer to the other endpoint then to any other point. If one of the adjacent edges is not in E then we have a direct contradiction to the assumption that σ is valid: in a valid color assignment any point has at most as many colors as incident edges (Lemma 5.1). If one of e's endpoints is closer to the other endpoint then all other edges, it would mean that e appears in both colors in E, a contradiction to the assumption that σ is valid. Then e is a local maximum and since σ is valid it must be that $p_a \circ\!\!\xrightarrow[\sigma]{1} p_{a-1}$, $p_a \circ\!\!\xrightarrow[\sigma]{2} p_b$, $p_b \circ\!\!\xrightarrow[\sigma]{2} p_a$, and $p_b \circ\!\!\xrightarrow[\sigma]{1} p_{b+1}$ (or with 1 and 2 swapped). Since e is a local maximum we know that the big gap $\{p_i, p_{i+1}\}$ is to the left or to the right of the edges adjacent to e. More precisely, we know that $p_{b+1} \leq p_i$ or $p_{i+1} \leq p_{a-1}$. But then in σ' the points from p_{a-1} to p_{b+1} either all have the same colors as in σ or they all have colors 1 and 2 swapped. Then the situation around e is the same in σ' as in σ, meaning that e only appears once. This obviously contradicts our assumption that e appears twice and shows that it is not possible for an edge to appear twice in E'.

Edge e appears in E' but not in E. Now, assume that there is an edge $e = \{p_a, p_b\} \in E'_1 \cup E'_2 \setminus (E_1 \cup E_2)$ with $a < b$.

If $b \leq i$ then $\sigma'(p_a) = \sigma(p_a)$ and $\sigma'(p_b) = \sigma(p_b)$. Assume without loss of generality that $e \in E'_1$. We then know that p_a connects to the same point with color 1 with respect to σ' as with respect to σ. Since we assume that $e \notin E$ it cannot be that $p_a \circ\!\!\xrightarrow[\sigma]{1} p_b$ and thus it also is true that $p_a \circ\!\!\not\xrightarrow[\sigma']{1} p_b$. It must then be the case that $p_b \circ\!\!\xrightarrow[\sigma']{1} p_a$ but $p_b \circ\!\!\not\xrightarrow[\sigma]{1} p_a$. Since σ is valid, it must be that $p_b \circ\!\!\xrightarrow[\sigma]{1} p_{b+1}$ which means that p_b is closer to p_{b+1} than to p_a. In order for $p_b \circ\!\!\xrightarrow[\sigma']{1} p_a$ to happen and not $p_b \circ\!\!\xrightarrow[\sigma']{1} p_{b+1}$ it must be that $1 \notin \sigma'(p_{b+1})$ and thus $\sigma'(p_{b+1}) \neq \sigma(p_{b+1})$. This can only be if $b + 1 > i$ and since $b \leq i$ it follows that $b = i$ and $b + 1 = i + 1$. But $\{p_b, p_{b+1}\} = \{p_i, p_{i+1}\}$ is a big gap

↪

and thus if $p_b \circ\!\xrightarrow[\sigma]{1} p_{b+1}$ is true then this gap would be present in E contradicting the assumption that σ is valid.

If $a > i$ we can apply the symmetric arguments and conclude that this is impossible, as well.

Finally, if $a \leq i$ and $b > i$ let $c \in \{1, 2\}$ be the color such that $e \in E'_c$. We know that $\sigma'(p_i) = \sigma'(p_{i+1}) = \{1\}$. If $c = 1$ then it must be that $p_a = p_i$ and $p_b = p_{i+1}$: If $a < i$ or $b > i + 1$ then there is a point with color 1 between p_a and p_b and $e \notin E'_1$, a contradiction. Since σ is valid it is the case that $1 \in \sigma(p_{a-1})$ and $2 \in \sigma(p_{b+1})$. As a result $1 \in \sigma'(p_{a-1})$ and $1 \in \sigma'(p_{b+1})$ but then, since e is a big gap, we have that $p_a \circ\!\xrightarrow[\sigma']{1} p_{a-1}$ and $p_b \circ\!\xrightarrow[\sigma']{1} p_{b+1}$. This is a contradiction to the assumption that $e \in E'_1$.

If $c = 2$ we know that $a < i$ and $b > i + 1$ since $\sigma'(p_i) = \sigma'(p_{i+1}) = \{1\}$. However, since $\sigma'(p_a) = \sigma(p_a)$ it would mean that $p_a \circ\!\xrightarrow[\sigma]{2} p_{i+1}$ since $\sigma(p_{i+1}) = \{2\}$ and p_{i+1} is between p_a and p_b. Then there is an illegal edge $\{p_a, p_{i+1}\} \in E$ contradicting the assumption that σ is valid.

Edge e appears in E but not in E′. Lastly, assume that there is an edge $e = \{p_a, p_{a+1}\} \in E_1 \cup E_2 \setminus (E'_1 \cup E'_2)$. Additionally, assume that c is the color such that $e \in E_c$.

If $a + 1 \leq i$ then we have that $\sigma'(p_a) = \sigma(p_a)$ and $\sigma'(p_{a+1}) = \sigma(p_{a+1})$. Then, if $p_a \circ\!\xrightarrow[\sigma]{c} p_{a+1}$ it follows that $p_a \circ\!\xrightarrow[\sigma']{c} p_{a+1}$ since there is no difference between σ and σ' to the left of p_a. This is a contradiction to the assumption that $e \notin E'$. If $p_{a+1} \circ\!\xrightarrow[\sigma]{c} p_a$, assume that $p_{a+1} \circ\!\not\xrightarrow[\sigma']{c} p_a$ which means that $p_{a+1} \circ\!\xrightarrow[\sigma']{c} p_b$ for some $p_b > p_{a+1}$. It must then be that $\sigma'(p_b) \neq \sigma(p_b)$ which can only be the case if $b > i$. If $c = 1$ then there can be two situations: If $a + 1 < i$ then we know that $1 \in \sigma(p_i)$, but then it would be that $b \leq i$, a contradiction. If $a + 1 = i$, we know that p_{a+1} is closer to p_a then to any other point since $\{p_i, p_{i+1}\}$ is a big gap. This would mean that $e \in E'_c$, a contradiction. In case $c = 2$ we know that $2 \notin \sigma'(p_{i+1})$ which means that $b > i + 1$. But, since $2 \in \sigma(p_{i+1})$ the assumption that $p_{a+1} \circ\!\xrightarrow[\sigma']{c} p_b$ would imply that $p_{a+1} \circ\!\xrightarrow[\sigma]{c} p_{i+1}$ since $p_{a+1} < p_{i+1} < p_b$. This would mean that $e \notin E$, a contradiction.

We have shown that it is impossible for an edge to appear twice in $E'_1 \cup E'_2$ or that there is an edge that is only in $E_1 \cup E_2$ but not in $E'_1 \cup E'_2$ or vice versa. Thus, we can conclude that $E_1 \cup E_2 = E'_1 \cup E'_2$ which means that $E = E'$ and this that σ' is valid. $\square$

We have now seen that a big gap can always be colored with the same color for both endpoints. At the same time, we know that small gaps need to have different colors for their endpoints.

6.3.3 Basic Color Assignments

As a result of the previous two sections we can see that the only edges that need some special consideration are the local maxima and the small gaps. For all other neighbor edges we have seen that it is sufficient if both endpoints have a common color for the edge to exist or, in the case of a big gap, not exist. This means that we will often talk about whether an edge from the neighbor edges is a local maximum or a small gap with respect to an input edge set. To make it easier we provide the following

▷ special edges

Definition 6.5. Let $E \subseteq E_P$ be an input edge set, $\mathscr{M}_E$ its set of local maxima, and $\mathscr{G}_E^-$ its set of small gaps. Any edge from E that is either a local maximum or a small gap is called a *special edge*. All those edges combined form the set of *special edges* in E, defined as $\mathscr{S}(E) = \mathscr{M}_E \cup \mathscr{G}_E^-$, also referred to by $\mathscr{S}_E$.

We can make an easy observation on the number of the special edges:

Observation 6.3. *Let $E \subseteq E_P$ be an input edge set for an input point set P with n points and $\mathscr{S}_E$ be the set of special edges. Then $2|\mathscr{S}_E| + 2 \leq n$ or put differently $\mathscr{S}_E \leq \lfloor n/2 \rfloor - 1$.*

Proof. A local maximum needs to have an edge to both its left and right. These two edges must be shorter than the local maximum and thus cannot be local maxima themselves. Similarly, a small gap needs to have an edge on both its sides and since those edges are then adjacent to a gap they cannot be local maxima either. Thus, between two special edges there must be a normal edge and the leftmost and rightmost edge in E cannot be a special edge.

Then both outermost points from P are not endpoints of special edges. Additionally, every point can be the endpoint of at most one special edge while every special edge needs two endpoints. We thus have at most $\lfloor n/2 - 2 \rfloor = \lfloor n/2 \rfloor - 1$ special edge. □

For an input edge set the corresponding input point set can be partitioned into the endpoints of special edges and all other points. We can group those other points together that are (transitively) adjacent: two points are in the same group if they don't have a special edge between them. We define this more formally as

▷ exclusive blocks
▷ blocks

Definition 6.6. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ with $u_j = \{p_{a_j}, p_{a_j+1}\}$ for all $1 \leq j \leq m$ be the set of special edges sorted from left to right.

We then define $m + 1$ *exclusive blocks* B_j^* each containing the points between two consecutive special edges, excluding the special edges' endpoints:

$$\begin{aligned} B_0^* &= \left\{p_i \in P \;\middle|\; p_i < p_{a_1}\right\}, \\ B_j^* &= \left\{p_i \in P \;\middle|\; p_{a_j+1} < p_i < p_{a_{j+1}}\right\} \quad \text{for all } 1 \leq j < m, \text{ and} \\ B_m^* &= \left\{p_i \in P \;\middle|\; p_{a_m+1} < p_i\right\}. \end{aligned}$$

↪

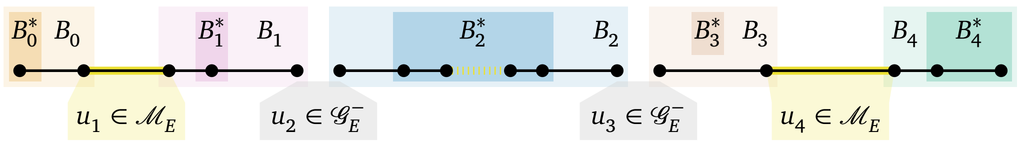


Figure 6.7: An input edge set and its corresponding blocks. There are four special edges u_1 to u_4 and thus five (exclusive) blocks. Note that B_3^* is empty since the right endpoint of u_3 and the left endpoint of u_4 are neighbors.

We also define $m + 1$ *blocks* B_j which include their encompassing special edges' closest endpoints:

$$\begin{aligned} B_0 &= B_0^* \cup \left\{ p_{a_1} \right\}, \\ B_j &= B_j^* \cup \left\{ p_{a_j+1}, p_{a_{j+1}} \right\} \quad \text{for all } 1 \le j < m, \text{ and} \\ B_m &= B_m^* \cup \left\{ p_{a_m+1} \right\}. \end{aligned}$$

See Figure 6.7 for an example input edge set and its corresponding blocks. We can see here that big gaps are not important as one is contained in B_2^*. Furthermore, an exclusive block may be empty, as demonstrated by B_3^*.

We now use the blocks and the special edges to define a special kind of color assignment. This kind of color assignment will be the only kind that we will need to look at, as will be shown later.

Definition 6.7. Let $E \subseteq E_P$ be an input edge set and $B_0, \dots, B_m$ be the corresponding blocks. Let $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a color assignment. We call σ a *basic color assignment (BCA)* with respect to E if it fulfills the following criteria:

▷ basic color assignment (BCA)

1. For all blocks B_j all points in it have a common color, i.e., there exists a $c_j \in C_{\hat{c}}$ such that $c_j \in \sigma(p)$ for all $p \in B_j$.
2. For two adjacent blocks B_j and B_{j+1} the colors c_j and c_{j+1} shared by their points are different.
3. For every local maximum $u_i \in \mathscr{S}_E \cap \mathscr{M}_E$ one endpoint has the two colors c_{i-1} and c_i of the blocks B_{i-1} and B_i incident to u_i. The other endpoint has only the color c_{i-1} or c_i of the block it is contained in.
4. Any point that is not an endpoint of a local maximum has only one color, the color of the block it is contained in.

Since the input edge set E will be generally clear from the context, we will mostly just say that σ is a *basic color assignment (BCA)*.

With this definition we can then show that if there exists a valid color assignment for an input edge set with the given number of colors, then there exists a valid BCA with the same number of colors:

Lemma 6.5. *Let $P \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, and $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a valid color assignment. Then there exists a valid BCA $\sigma' : P \to \mathbb{C}_{\hat{c}}$.*

Proof. If we apply Lemma 6.3 exhaustively we transform σ into a valid color assignment σ^* in which every point except one endpoint of each local maximum has exactly one color; for each local maximum exactly one endpoint has two colors. We can then apply Lemma 6.4 exhaustively to transform σ^* into another valid color assignment σ' where for every big gap its endpoints have the same color. Since Lemma 6.4 does not change the *number* of colors of any point, it thus still holds that no point has two colors except one endpoint of each local maximum. It remains to show that σ' fulfills all criteria to be a BCA.

For criterion 1 assume that there is a block B_j and two points $p_i, p_k \in B_j$ with $p_i < p_k$ such that the two points don't have a common color. If there are multiple such pairs of points, we choose one with the least amount of points between them. Then it must be that $p_k = p_{i+1}$: No point p_l with $p_i < p_l < p_k$ can have two colors. If p_l existed and didn't share a color with p_i or p_k this would violate the assumption that p_i and p_k have as few points between them as possible, because both p_i and p_l as well as p_l and p_k would be closer pairs. If p_l existed and shared a color with either p_i or p_k we would have the same assumption violation. However, we now have two neighboring points p_i and p_{i+1} that don't have a common color. Thus, $e_i = \{p_i, p_{i+1}\} \notin \mathcal{N}(P, \sigma')$. If $e_i \in E$, this is a contradiction to the fact that σ' is valid. On the other hand, if $e_i \notin E$ it follows that e_i is a gap and because both endpoints are in the same block it is a big gap. However, since Lemma 6.4 was applied exhaustively, it must be that all endpoints of big gaps have the same color, a contradiction. Thus, criterion 1 must hold for σ'.

For criterion 2 assume that there is a special edge $u_j = \{p_i, p_{i+1}\}$ such that for its two incident blocks B_{j-1} and B_j the shared colors c_{j-1} and c_j are the same. This means that $c_j \in \sigma'(p_i)$ and $c_j \in \sigma'(p_{i+1})$. If u_j is a small gap we know then that $p_i \overset{c_j}{\circ\!\!\rightarrow} p_{i+1}$ or $p_{i+1} \overset{c_j}{\circ\!\!\rightarrow} p_i$ which would mean that $u_j \in \mathcal{N}(P, \sigma')$, a contradiction since σ' is valid. If u_j is a local maximum we know that $p_i \overset{c_j}{\circ\!\!\rightarrow} p_{i-1}$ and $p_{i+1} \overset{c_j}{\circ\!\!\rightarrow} p_{i+2}$ and thus u_i does not appear in $\mathcal{N}(P, \sigma')$ with color c_j. However, since not both endpoints have a second color, u_i does not appear in $\mathcal{N}(P, \sigma')$ at all, a contradiction. As a result, criterion 2 must hold for σ'.

For criterion 3 we look at each local maximum $u_j = \{p_i, p_{i+1}\} \in \mathcal{S}_E \cap \mathcal{M}_E$. Let c_{j-1} and c_j be the shared colors with $c_{j-1} \neq c_j$ for u_j's incident blocks according to criteria 1 and 2. It follows that $c_{j-1} \in \sigma'(p_i)$ and $c_j \in \sigma'(p_{i+1})$. We have already observed at the beginning of this proof that for each local maximum exactly one endpoint has two colors and the other has one color. Let, without loss of generality, p_i be the endpoint

↪

with two colors. Since $u_j \in E$, it must be that $u_j \in \mathcal{N}(P, \sigma')$. For this to happen, p_i and p_{i+1} must have a common color. Since p_i is allowed exactly one addition color, this color must then be c_j, the color assigned to p_{i+1}. Thus, criterion 3 holds for σ'.

For criterion 4 we already noted that in σ' all points have exactly one color except for one endpoint of each local maximum that has two colors. Thus, it is clear that any point that is not the endpoint of a local maximum has only one color. Due to criterion 1 it must be the color shared with all other points in its block.

We have shown that σ' which is constructed by applying first Lemma 6.3 and then Lemma 6.4 exhaustively is valid and fulfills all criteria of Definition 6.7 and is thus a BCA. □

We now know that for every valid color assignment there exists a valid BCA and we can even transform the former into the latter. This lets us restrict ourselves to look for BCAs which will be much easier due to their restricted structure. We will now also see that for a given BCA we don't need to check all points and its connection(s) to see whether the BCA is valid. In fact, we will only need to check the endpoints of special edges as we will show in

Lemma 6.6. *Let $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ be an input edge set, $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a BCA, and let $E' = \mathcal{N}(P, \sigma)$ be the edges of the resulting CNNM. If σ is not valid, i.e., $E' \neq E$, then the following statements hold:*

1. *For all edges $e_i \in E$ with $e_i \notin E'$ we know that e_i is either a local maximum or directly to the left or right of a small gap, i.e., either $e_i \in \mathcal{M}_E$, $e_{i+1} \in \mathcal{G}_E^-$, or $e_{i-1} \in \mathcal{G}_E^-$.*
2. *For all edges $e \in E'$ that appear more than once in E' both e's endpoints are endpoints of local maxima.*
3. *For all edges $e = \{p_i, p_j\} \in E'$ such that $e \notin E$ there exists a color $c \in C_{\hat{c}}$ such that $p_i \overset{c}{\circ\!\!\rightarrow} p_j$ and p_i is the endpoint of a special edge. Furthermore, e is an illegal edge, i.e., $e \notin E_P$ and thus not a gap.*

Proof. We start with the first statement. Let $e_i = \{p_a, p_{a+1}\} \in E$ be an edge such that $e_i \notin E'$. Since σ is a BCA and e_i is not a gap there is a color $c \in C_{\hat{c}}$ such that $c \in \sigma(p_a)$ and $c \in \sigma(p_{a+1})$. Assume first that e_i is not next to a gap and not a local maximum. This assumption tells us that at least one of $\|e_{i-1}\| > \|e_i\|$ or $\|e_i\| < \|e_{i+1}\|$ holds. If, on the other hand, e_i is next to a big gap e_j with $j \in \{i-1, i+1\}$ then $\|e_j\| > \|e_i\|$. As a result $p_a \circ\!\!\rightarrow p_{a+1}$ or $p_{a+1} \circ\!\!\rightarrow p_a$ and since both points have a common color c it must be that $p_a \overset{c}{\circ\!\!\rightarrow} p_{a+1}$ or $p_{a+1} \overset{c}{\circ\!\!\rightarrow} p_a$. Thus, $e_i \in E'$, a contradiction. It must then be that e_i is either a local maximum or next to a small gap.

↪

We now show the second statement. If an edge appears more than once in E' both endpoints must have the same two colors. Since only endpoints of local maxima have two colors, the claim follows directly.

Finally, we show the last statement. If we have an edge $e = \{p_i, p_j\} \in E'$ with $i < j$ that does not appear in E, there are two possibilities. Either $e \in E_P$ which means that e is a gap in E or $e \notin E_P$ meaning it's an illegal edge. We first show that e cannot be a gap: It cannot be a small gap since in a BCA both endpoints of a small gap do not have a common color. If e were a big gap, then $j = i + 1$, and since p_i and p_j are in the same exclusive block it would be that p_{i-1}, p_i, p_{i+1}, and p_{i+2} all have a common color c. However, since e is a big gap, then $p_i \overset{c}{\hookrightarrow} p_{i-1}$ and $p_j \overset{c}{\hookrightarrow} p_{j+1}$ and thus e would not be in E', a contradiction. As a result, e must be an illegal edge, i.e., $j < i - 1$ or $j > i + 1$. Assume, without loss of generality, that $p_i \overset{c}{\hookrightarrow} p_j$ for some $c \in C_{\hat{c}}$. Assume furthermore that p_i is not the endpoint of a special edge. Then $\sigma(p_i) = \{c\}$ as p_i has only one color and also both adjacent points will also have color c since σ is a BCA and p_i is in an exclusive block. But then p_i will connect to either p_{i-1} or p_{i+1} contradicting the assumption that e is an illegal edge. □

With this result we will be able to more easily show that a given BCA is valid: We will assume that the BCA is not valid and we then know that the only edges and points we need to check are those described in Lemma 6.6. We will now directly use this lemma to give an exact characterization for when a BCA is valid and when it is not:

Lemma 6.7. *Let $E \subseteq E_P$ be an input edge set for an input point set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$. Let $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a BCA. Then σ is valid if and only if the following conditions hold:*

1. *For all small gaps $e_i = \{p_i, p_{i+1}\} \in \mathscr{G}_E^-$, let $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_r\}$ be its endpoints' colors. Then, for all points $p \in P \cap \overleftarrow{R}_E(e_i)$ it holds that $c_r \notin \sigma(p)$ and for all points $p \in P \cap \overrightarrow{R}_E(e_i)$ it holds that $c_l \notin \sigma(p)$.*

2. *For all local maxima $e_i = \{p_i, p_{i+1}\} \in \mathscr{M}_E$ with two colors for the right endpoint, let $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_l, c_r\}$ be the endpoints' colors. Then, for all points $p \in P \cap \overrightarrow{R}_E(e_i)$ it holds that $c_l \notin \sigma(p)$.*

3. *For all local maxima $e_i = \{p_i, p_{i+1}\} \in \mathscr{M}_E$ with two colors for the left endpoint, let $\sigma(p_i) = \{c_l, c_r\}$ and $\sigma(p_{i+1}) = \{c_r\}$ be the endpoints' colors. Then, for all points $p \in P \cap \overleftarrow{R}_E(e_i)$ it holds that $c_r \notin \sigma(p)$.*

Proof. We first show that if all three conditions hold, then σ is valid. We do this by showing that if σ is not valid then at least one of the conditions does not hold. Let $E' = \mathscr{N}(P, \sigma)$ be the edges of the CNNM for σ. It must then be that there is an edge $e_i \in E \setminus E'$, an edge that appears multiple times in E' (i.e., E' is a multiset), or an edge $e \in E' \setminus E$. We can use Lemma 6.6 to see that we can restrict ourselves to checking specific edges or points in all three cases.

↪

Assume first that there is an edge $e_i = \{p_i, p_{i+1}\} \in E \setminus E'$. Statement 1 of Lemma 6.6 tells us that e_i is either a local maximum or adjacent to a small gap. If e_i is a local maximum we know that either $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_l, c_r\}$ or $\sigma(p_i) = \{c_l, c_r\}$ and $\sigma(p_{i+1}) = \{c_r\}$ for two distinct colors $c_l, c_r \in C_{\hat{c}}$. In the first case it cannot be that $p_{i+1} \overset{c_l}{\circ\!\!\longrightarrow} p_i$ since $e_i \notin E'$. It must then be that there is a point $p_k > p_{i+1}$ such that $p_{i+1} \overset{c_l}{\circ\!\!\longrightarrow} p_k$. But then both $c_l \in \sigma(p_k)$ and $p_k \in \overrightarrow{R}_E(e_i)$ must be true, and thus condition 2 does not hold. In the second case it cannot be that $p_i \overset{c_r}{\circ\!\!\longrightarrow} p_{i+1}$ since $e_i \notin E'$. It must then be that there is a point $p_k < p_i$ such that $p_i \overset{c_r}{\circ\!\!\longrightarrow} p_k$. But then both $c_r \in \sigma(p_k)$ and $p_k \in \overleftarrow{R}_E(e_i)$ must be true, which means that condition 3 does not hold.

If e_i is adjacent to a small gap, assume first that $e_{i-1} \in \mathscr{G}_E^-$ is this gap. We then know that there are two distinct colors $c_l, c_r \in C_{\hat{c}}$ such that $\sigma(p_{i-1}) = \{c_l\}$ and $\sigma(p_i) = \{c_r\}$. Since σ is a BCA and p_i and p_{i+1} are in the same block, it follows that $c_r \in \sigma(p_{i+1})$. Due to our assumption it cannot be that $p_i \overset{c_r}{\circ\!\!\longrightarrow} p_{i+1}$ which means that there must be a $p_k < p_i$ such that $p_i \overset{c_r}{\circ\!\!\longrightarrow} p_k$. But then both $c_r \in \sigma(p_k)$ and $p_k \in \overline{R}(p_i, p_{i+1})$ must be true and by definition $\overleftarrow{R}_E(e_i) = \overline{R}(p_i, p_{i+1})$. Then condition 1 does not hold. If e_{i+1} and not e_i, as assumed, is the small gap, the symmetric arguments also show that condition 1 does not hold.

Assume now that there is an edge $e = \{p_i, p_j\} \in E'$ that appears more than once in E'. Statement 2 of Lemma 6.6 tells us that both endpoints are endpoints of local maxima. Additionally, it must be that $\sigma(p_i) = \sigma(p_j) = \{c_1, c_2\}$ for two distinct colors $c_1, c_2 \in C_{\hat{c}}$. Let without loss of generality be that $p_i < p_j$. We now show that it must be the case that $p_j = p_{i+1}$. Assume that $p_j > p_{i+1}$. We know that, since σ is a BCA, p_{i+1} must have either c_1 or c_2 assigned to it; let it be c_1, without loss of generality. But then $p_i \overset{c_1}{\circ\!\!\longrightarrow} p_{i+1}$ and because $p_i < p_{i+1} < p_j$ it would be the case that p_j connected with color c_1 to p_{i+1} or possibly a point between p_{i+1} and p_j; certainly not p_i. Thus, e would not be present in E' in color c_1, a contradiction. Now that we know that $p_j = p_{i+1}$ we know that e_{i-1} and e_{i+1} are local maxima and thus p_i and p_{i+1} are the right and left endpoint of their local maximum, respectively. This also means that p_i and p_{i+1} are closer to each other than to their respective other endpoints of their local maxima. Let, without loss of generality, $\sigma(p_{i-1}) = \{c_1\}$. Then, it is clear that $p_{i+1} \in \overrightarrow{R}_E(e_{i-1})$ and since $c_1 \in \sigma(p_{i+1})$ condition 2 does not hold. Looking at p_{i+1} we see with symmetric arguments that condition 3 does not hold either.

Lastly, assume that there is an edge $e = \{p_i, p_j\} \in E'$ such that $e \notin E$. Let $c \in C_{\hat{c}}$ be the color in which e appears in E' and assume without loss of generality that $p_i \overset{c}{\circ\!\!\longrightarrow} p_j$. Statement 3 of Lemma 6.6 tells us that then e is an illegal edge and p_i is the endpoint of a special edge $u_k \in \mathscr{S}_E$. Let, without loss of generality, p_i be the left endpoint of u_k and thus $u_k = e_i$.

If u_k is a local maximum, first assume that p_i is the endpoint with two colors, which means that $\sigma(p_i) = \{c_l, c_r\}$, $c_l \in \sigma(p_{i-1})$, and $\sigma(p_{i+1}) = \{c_r\}$ for two distinct colors $c_l, c_r \in C_{\hat{c}}$. Then it cannot be that $c = c_l$ because $p_i \overset{c_l}{\circ\!\!\longrightarrow} p_{i-1}$ since p_i is closer to p_{i-1}

↪

than to any other point and p_j is not p_{i-1} since e is an illegal edge. Thus, it must be that $c = c_r$ which means that p_j is to the left of p_i, i.e., $p_j < p_{i-1}$. But then p_i must be closer to p_j than to p_{i+1} and p_j must have color c_r. This means that $p_j \in \overleftarrow{R}(p_i, p_{i+1})$ which is $\overleftarrow{R}_E(e_i)$ by definition, and together with $c_r \in \sigma(p_j)$ this shows that condition 3 does not hold. If p_i is the endpoint with one color, we know that $\sigma(p_i) = \{c_l\}$, $c_l \in \sigma(p_{i-1})$, and $\sigma(p_{i+1}) = \{c_l, c_r\}$ for two distinct colors $c_l, c_r \in C_{\hat{c}}$. Since c_l is the only color assigned to p_i, it must be that $c = c_l$. However, $p_i \overset{c_l}{\circ\!\!\rightarrow} p_{i-1}$ since p_i is closer to p_{i-1} than to p_{i+1} which means that $p_j = p_{i-1}$. This is a contradiction to our assumption that e is an illegal edge. If p_i were the right endpoint of u_k, the symmetric arguments would show that condition 2 does not hold.

If u_k is a small gap, remember that we assume that p_i is its left endpoint, and let $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_r\}$ be the gap's endpoints' colors. Then p_j cannot be to the left of p_i since p_i would always connect with color c_l to p_{i-1} instead of p_j. Additionally, $p_j = p_{i-1}$ is not possible since e is an illegal edge, and thus $p_j > p_{i+1}$ must hold. In order for $p_i \overset{c_r}{\circ\!\!\rightarrow} p_j$ to happen it must be that $c_r \in \sigma(p_j)$ and $p_j \in \overrightarrow{R}(p_i, p_{i-1}) = \overrightarrow{R}_E(e_i)$ which shows that condition 1 does not hold. The symmetric arguments hold if p_i were the right endpoint of u_k.

We have now shown that in all cases which may make σ not valid we can conclude that at least one of the three conditions does not hold. As a result we can conclude that if all three conditions hold that then σ is valid.

We now show that if σ is valid then all three conditions hold.

Look at all small gaps $e_i = \{p_i, p_{i+1}\} \in \mathcal{G}_E^-$ and let $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_r\}$ the endpoints' colors. Since σ is a BCA we know that $c_l \neq c_r$ and since σ is valid it must be that $p_i \overset{c_l}{\circ\!\!\rightarrow} p_{i-1}$ and $p_{i+1} \overset{c_r}{\circ\!\!\rightarrow} p_{i+2}$. This is only possible if there is no point to the right of p_i with color c_l to which p_i is closer than to p_{i-1}. Or in other words, no point inside $\overrightarrow{R}_E(e_i)$ can have color c_l. The same holds for p_{i+1}, $\overleftarrow{R}_E(e_i)$ and color c_r. Thus, condition 1 holds.

Look at all local maxima $e_i = \{p_i, p_{i+1}\} \in \mathcal{M}_E$ with two colors for the right endpoint and let $\sigma(p_i) = \{c_l\}$ and $\sigma(p_{i+1}) = \{c_l, c_r\}$ be the endpoints' colors. We then know that $c_l \in \sigma(p_{i-1})$ since σ is a BCA and p_{i-1} and p_i are in the same block. Since e_i is a local maximum, it follows that $p_i \overset{c_l}{\circ\!\!\rightarrow} p_{i-1}$. Then it must be that $p_{i+1} \overset{c_l}{\circ\!\!\rightarrow} p_i$ as otherwise $e_i \notin \mathcal{N}(P, \sigma)$ which would contradict the assumption that σ is valid. But then there can be no point with color c_l inside $\overrightarrow{R}_E(e_i)$, showing that condition 2 holds.

With symmetric arguments we can see that condition 3 holds for all local maxima with two colors for its left endpoint.

We have shown that all three conditions hold if σ is valid, which concludes the proof. $\square$

With Lemma 6.7 we now have a powerful tool to determine whether a BCA is valid. We only need to check whether (a subset of) the restricted regions of each special edge contain other points with a forbidden color. This also tells us that the 1D-CNNG-Gaps problem (and thus also the 1D-CNNG problem) is contained in NP: Given an input point set and an input edge set we know that any BCA has linear size in the input. In addition, using Lemma 6.7, we can check whether the BCA is valid in polynomial time. With this knowledge for the general case we will now start tackling the special case for $\hat{c} = 2$.

6.4 Two Colors

After the previous structural observations we will now see which inputs can be successfully colored if we are allowed to use two colors. From Lemma 6.5 we know that we only need to look for BCAs. In order to find a BCA we then only need to decide for each local maximum in the input edge set whether its left or right endpoint should have two colors. All colors for the other points are then already determined since with two colors, the colors of the blocks have to alternate. Assuming that the input edge set has k local maxima, there are 2^k different BCAs (or 2^{k+1} if we count those with swapped colors) that we would need to check whether they result in a CNNG. Since the number of local maxima can be linear in the number of input points (remember Observation 6.3), we want to find a fast algorithm.

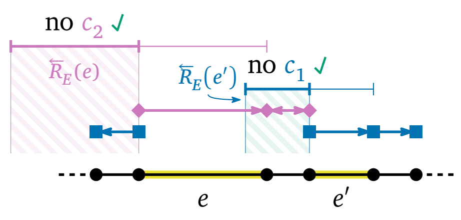
6.4.1 Only Adjacent Color Changes Are Important

We will now see that, luckily, we do not need to check all BCAs. For two colors every odd-numbered block will have one color while every even-numbered block will have the other color. As a result, every special edge will have one endpoint with one color and one endpoint with the other color (for local maxima one endpoint even has both colors). In addition, the endpoints of special edges are the only points at which a color changes when going from left to right (or right to left) along the points. This leads to the following

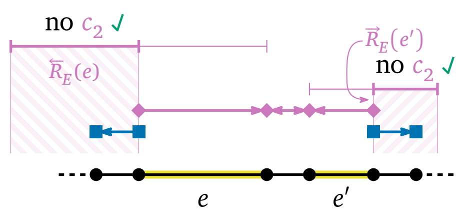
Lemma 6.8. *Let $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ an input edge set, $\mathscr{S}_E = \{u_1, \ldots, u_m\}$ be the special edges, and $\sigma : P \to \mathbb{C}_2$ be a BCA with two colors. For every special edge u_i let $c_{i,l}$ be the color shared among block B_{i-1} to the left of u_i and $c_{i,r}$ be the color shared among block B_i to the right of u_i. Then both following statements hold:*

1. *For all u_i with $i < m$ the closest point to the right of u_i with color $c_{i,l}$ is the endpoint of u_{i+1} with two colors if u_{i+1} is a local maximum or the right endpoint of u_{i+1} if it is a small gap.*

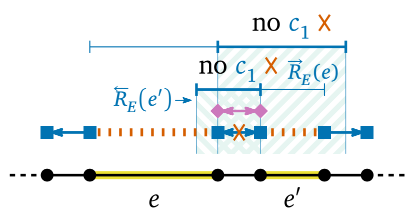
2. *For all u_i with $i > 1$ the closest point to the left of u_i with color $c_{i,r}$ is the endpoint of u_{i-1} with two colors if u_{i-1} is a local maximum or the left endpoint of u_{i-1} if it is a small gap.*



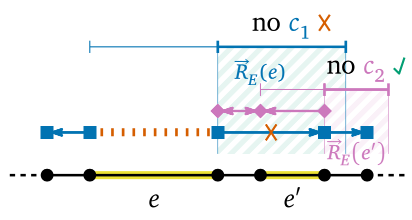
(a) Both left endpoints can have two colors, as the left endpoint of e is not inside $\overleftarrow{R}_E(e')$.



(b) Two colors for the left endpoint of e and right endpoint of e' is always possible.



(c) Both restricted regions contain the endpoint with two colors.



(d) $\overrightarrow{R}_E(e)$ contains both endpoints of e', even the right one with two colors.

Figure 6.8: The four possible ways to color the two highlighted local maxima e and e' with two colors. The color assignment works if none of the shown restricted regions contain the point with two colors of the other local maximum.

Proof. We will show statement 1 and then statement 2 follows due to symmetry. If $i < m$ we know that the color shared in B_i is $c_{i,r}$ and then the color shared in B_{i+1} is $c_{i+1,r} = c_{i,l}$ again, since we have only two alternating colors. No point in B_i^* has color $c_{i,l}$ since they all have only one color. The right endpoint of u_{i+1} definitely has color $c_{i,l}$ assigned to it, since it is the leftmost point in B_{i+1}. If u_{i+1} is a small gap or its right endpoint has two colors, then u_{i+1}'s left endpoint has only color $c_{i,r} = c_{i+1,l}$; thus u_{i+1}'s right endpoint is the leftmost one with $c_{i,l}$. Only if u_{i+1} is a local maximum whose left endpoint has two colors, then its left endpoint also has color $c_{i,l} = c_{i+1,r}$ assigned to it and is thus the leftmost points with this color. ☐

With Lemmas 6.7 and 6.8 we can already infer that we need to check only few conditions in order to determine whether a given BCA is valid: If we have a small gap $u_i \in \mathscr{S}_E \cap \mathscr{G}_E^-$ we need to ensure that the right endpoint of u_{i+1} is not inside $\overrightarrow{R}_E(u_i)$. In the case that u_{i+1} is a local maximum with two colors for its left endpoint we also need to ensure that its left endpoint is not inside $\overrightarrow{R}_E(u_i)$. The mirrored checks apply for u_{i-1}. If we have a local maximum $u_i \in \mathscr{S}_E \cap \mathscr{M}_E$ and we have two colors for its right endpoint, we also need to ensure that the right endpoint of u_{i+1} is not inside $\overrightarrow{R}_E(u_i)$. In the case that u_{i+1} is a local maximum with two colors for its left endpoint we also need to ensure that its left endpoint is not inside $\overrightarrow{R}_E(u_i)$. If we have two colors for its left endpoint, we apply the mirrored versions of the checks.

We can see examples of the situations between two adjacent local maxima e and e' in Figure 6.8. To describe those, we will use the term *inside* or *outside* endpoint to refer to the local maximum's endpoint that is closer or further away from the other local maximum, respectively. Here we look at the four possible combinations to assign two colors to either the left or the right endpoint of each local maximum. In this situation it is possible to assign two colors to both left endpoints (Figure 6.8a) since e' is short enough that its left restricted region does not contain the left endpoint of e. It is, of course always possible to assign two colors to outside endpoints of the local maximum (Figure 6.8b) as the restricted regions face away from each other. Giving both inside endpoints two colors (Figure 6.8c) can work if the local maxima are further away from each other. In this situation, however, both restricted regions contain the other inside endpoint. As a result, the corresponding CNNM has both local maxima missing and the edge between them appears twice. Finally, giving both right endpoints two colors (Figure 6.8d) does not work in this situation as the right restricted region of e contains both endpoints of e' and thus also the right endpoint with two colors. The corresponding CNNM lacks e but includes an illegal edge between both right endpoints of e and e'.

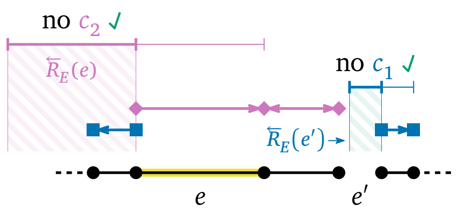
In Figure 6.9 we can then see different situations between a local maximum e and a small gap e'. For the small gap we have three different situations: One where its restricted region does not contain any endpoints of e (Figures 6.9a and 6.9b), one where it contains one of the two endpoints (Figures 6.9c and 6.9d), and one where it contains both endpoints (Figures 6.9e and 6.9f). In the first two situations (Figures 6.9a and 6.9c) choosing e's left endpoint for two colors works but in the last situation (Figure 6.9e) it does not since the left restricted region of e' contains both endpoints. For choosing e's right endpoint for two colors, we see that it does not work in the last two situations (Figures 6.9d and 6.9f) since it is contained in the left restricted region of e'. In the first situation (Figure 6.9b) it does not work in this situation since e's right restricted region contains both endpoints of e'.

Finally, in Figure 6.10 we can observe two different situations for two small gaps next to each other. Here we only need to ensure that the restricted region does not contain the outside endpoint of the other small gap. Thus, there are generally only two situations. In Figure 6.10a we have the situation in which a restricted region contains both endpoints of the other small gap and thus an illegal edge is created and the edge next to the small gap is missing. In Figure 6.10b, on the other hand, we see a valid solution since no outside endpoint is contained in the restricted regions.

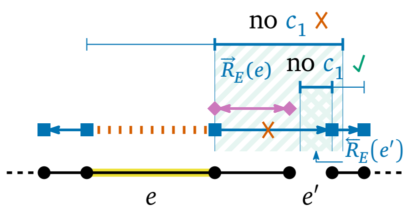
With the information on which BCAs are valid and how to check it we can now start with the first steps towards the actual algorithm.

6.4.2 The Realization Graph

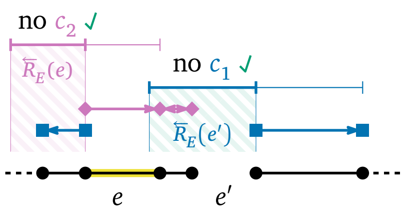
We now know from Lemma 6.7 that to decide whether a given BCA is valid, we only need to check whether the relevant restricted regions of all special edges contain a



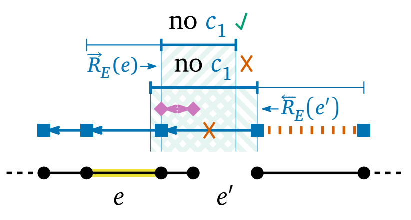
(a) No restricted region contains any problematic points.



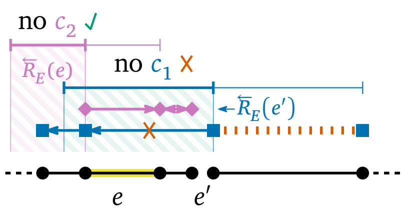
(b) $\overrightarrow{R}_E(e)$ contains both endpoints of e', resulting in an illegal edge and e missing.



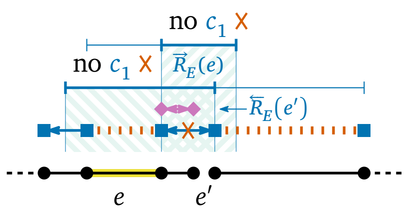
(c) $\overleftarrow{R}_E(e')$ contains the right endpoint of e, but it has only one color.



(d) $\overleftarrow{R}_E(e')$ contains the right endpoint of e, creating the wrong edge.

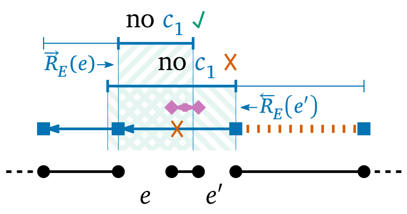


(e) $\overleftarrow{R}_E(e')$ contains both endpoints of e, resulting in the wrong edge.

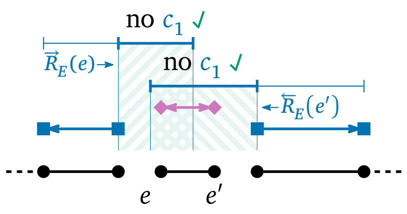


(f) $\overleftarrow{R}_E(e')$ contains both endpoints of e, resulting in the wrong edge.

Figure 6.9: A small gap e' may be in conflict with a neighboring local maximum e. (a) and (b): If $\overleftarrow{R}_E(e')$ does not contain any endpoint of e, it depends on the size of e which choice of endpoint is valid. (c) and (d): If $\overleftarrow{R}_E(e')$ contains one endpoint of e, then this endpoint cannot have two colors. (e) and (f): If $\overleftarrow{R}_E(e')$ contains both endpoints of e, then no choice for e is valid.



(a) The left restricted region of e' contains both endpoints of e.



(b) No restricted region of e or e' contains both endpoints of the other small gap.

Figure 6.10: Two neighboring small gaps may interfere if a restricted region of one gap contains both endpoints of the other gap, as seen in (a). If this is not the case then we have a valid solution, see (b).

point with the wrong color. By Lemma 6.8 for every special edge the points which have the wrong color are endpoints of the adjacent special edges. In the last section we also saw many examples of how adjacent special edges and their color assignments interact.

Given an input edge set $E \subseteq E_P$ we will now create a directed acyclic graph for E which we call the *realization graph* $H = (V, A)$. The graph will model all possible BCAs and its creation will be the biggest part of the upcoming algorithm to find a BCA with respect to E with two colors. Let $\mathcal{S}_E = \{u_1, \ldots, u_m\}$ be the special edges. Then, each $u_i \in \mathcal{S}_E$ has a node in H for every way the color can change at u_i. This means that each small gap has one node $\overleftrightarrow{u}_i$ while local maxima have two nodes $\overleftarrow{u}_i$ and $\vec{u}_i$. We also add a start node s and an end node t. Formally, we can define V as

$$V = \{s, t\} \cup \{\overleftrightarrow{u}_i \mid u_i \in \mathcal{G}_E^-\} \cup \bigcup_{u_i \in \mathcal{M}_E} \{\overleftarrow{u}_i, \vec{u}_i\}.$$

For the arcs A in H we first add all possible arcs from s to nodes corresponding to u_1 and from nodes corresponding to u_m to t. We then create certain arcs between nodes corresponding to adjacent special edges. As we have seen in Section 6.4.1 it is sufficient to check adjacent special edges to see whether the choice of endpoints of local maxima which are assigned two colors is valid. That is, the following conditions for when an arc is created are exactly the conditions from Lemma 6.7 with the knowledge from Lemma 6.8 that it suffices to check the endpoints of the adjacent special edges. We therefore look at all adjacent pairs $u_i = \{p_a, p_{a+1}\}$ and $u_{i+1} = \{p_b, p_{b+1}\}$ for all $1 \leq i < m$ and distinguish them by whether they are local maxima or small gaps:

- For the case that $u_i, u_{i+1} \in \mathcal{G}_E^-$ we observed that we can color with two colors as long as both restricted regions do not contain both endpoints of the other small gap. As a result we add the edge $(\overleftrightarrow{u}_i, \overleftrightarrow{u}_{i+1})$ if and only if $p_a \notin \overleftarrow{R}_E(u_{i+1})$ and $p_{b+1} \notin \vec{R}_E(u_i)$. Remember Figure 6.10 for two possible outcomes.

- If $u_i \in \mathscr{M}_E$ and $u_{i+1} \in \mathscr{G}_E^-$ we must ensure that the endpoint of u_i that changes color is not inside $\overleftarrow{R}_E(u_{i+1})$. Thus, we add $(\overleftarrow{u}_i, \overleftrightarrow{u}_{i+1})$ if $p_a \notin \overleftarrow{R}_E(u_{i+1})$, see Figure 6.9a for an example. Furthermore, if the right endpoint of u_i should have two colors, then u_i's right restricted region must not contain the right endpoint of u_{i+1}. Hence, we add $(\overrightarrow{u}_i, \overleftrightarrow{u}_{i+1})$ only if $p_{a+1} \notin \overleftarrow{R}_E(u_{i+1})$ and $p_{b+1} \notin \overrightarrow{R}_E(u_i)$. See Figure 6.9b for a situation in which the arc would not be added.

- If $u_i \in \mathscr{G}_E^-$ and $u_{i+1} \in \mathscr{M}_E$ we have the symmetric situation to the one before. Thus, add arc $(\overleftrightarrow{u}_i, \overrightarrow{u}_{i+1})$ if $p_{b+1} \notin \overrightarrow{R}_E(u_i)$, and add $(\overleftrightarrow{u}_i, \overleftarrow{u}_{i+1})$ only if $p_b \notin \overrightarrow{R}_E(u_i)$ and $p_a \notin \overleftarrow{R}_E(u_{i+1})$.

- If $u_i, u_{i+1} \in \mathscr{M}_E$, we have four possible arcs between their respective nodes. The arc $(\overleftarrow{u}_i, \overrightarrow{u}_{i+1})$ is always added, see Figure 6.8b for a depiction. For the other arcs we need to check the relevant restrictions: If $p_{b+1} \notin \overrightarrow{R}_E(u_i)$ we can add the arc $(\overrightarrow{u}_i, \overrightarrow{u}_{i+1})$; see Figure 6.8d for a situation in which the arc could not be added. If $p_a \notin \overleftarrow{R}_E(u_{i+1})$ we can add the arc $(\overleftarrow{u}_i, \overleftarrow{u}_{i+1})$; see Figure 6.8a for a situation in which the arc could be added. The last arc $(\overrightarrow{u}_i, \overleftarrow{u}_{i+1})$ is added only if $p_b \notin \overrightarrow{R}_E(u_i)$ and $p_{a+1} \notin \overleftarrow{R}_E(u_{i+1})$; see Figure 6.8c for a depiction of a situation in which the arc would not be added.

We now have a realization graph for a specific input edge set. In this graph we may be able to find *s*-*t* paths, i.e., paths that start at the source *s* and end at the sink *t*. For every such path Q we first define a color assignment, show that it is a BCA, and then show that it is valid for E.

▷ corresponding color assignment

Definition 6.8. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ an input edge set, $H = (V, A)$ be the corresponding realization graph, $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the special edges, and $B_0, \dots, B_m$ be the blocks for E. Let furthermore, Q be an *s*-*t* path in H. We then define the *corresponding color assignment* $\sigma_Q : P \to \mathbb{C}_2$ as follows.

For all $0 \le i \le m$ the points in block B_i are assigned the color set $\{(i \bmod 2) + 1\}$. This means that $\sigma_Q(p) = \{(i \bmod 2) + 1\}$ for all $p \in P$ if $p \in B_i$. As we can see, this part is independent of Q.

We now only need to change some points to have both colors. For every local maximum $\{p_k, p_{k+1}\} = u_i \in \mathscr{S}_E \cap \mathscr{M}_E$, if Q contains $\overleftarrow{u}_i$ then set $\sigma_Q(p_k) = \{1, 2\}$, and if Q contains $\overrightarrow{u}_i$ then set $\sigma_Q(p_{k+1}) = \{1, 2\}$.

From its definition it is relatively easy to see that σ_Q is a BCA:

Observation 6.4. *Let $P \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ an input edge set, $H = (V, A)$ be the corresponding realization graph, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the special edges. Let furthermore, Q be an s-t path in H and σ_Q the corresponding color assignment. Then σ_Q is a BCA.*

Proof. We check the four criteria from Definition 6.7 individually: It is clear that criteria 1 and 2 are met by definition. For criterion 4 we can see that the second part of σ_Q's definition only applies for endpoints of local maxima; thus all other points do not get a second color. It is also clear that criterion 3 follows directly if we can ensure that for each local maximum u_i a path Q will contain exactly one of $\overleftarrow{u}_i$ and $\overrightarrow{u}_i$. Due to its construction H only contains arcs from s to nodes corresponding to u_1, arcs from nodes corresponding to u_i to nodes corresponding to u_{i+1} for all $1 \leq i < m$, and arcs from nodes corresponding to u_m to t. As a result, any s-t path in H must contain exactly one node for each special edge $u_i \in \mathscr{S}_E$, thus fulfilling criterion 3. □

It now remains to show that for every s-t path Q the corresponding BCA σ_Q is valid:

Lemma 6.9. *Let $E \subseteq E_P$ be an input edge set, and $H = (V, A)$ be the corresponding realization graph. Let Q be an s-t path in H and $\sigma_Q : P \to \mathbb{C}_2$ be the corresponding BCA. Then σ_Q is valid.*

Proof. As mentioned before, the combination of Lemma 6.7 and Lemma 6.8 tells us exactly which conditions we need to check for each special edge to determine whether a BCA is valid. That is, we need to check the conditions from Lemma 6.7, but it is sufficient to check the endpoints of the closest special edge on either side. These conditions are exactly those that are checked before adding an arc to H. Hence, an arc in H between two nodes exists only if the corresponding color assignment is valid when restricted to the corresponding special edges and the three blocks around and between them. □

After seeing that every s-t path corresponds to a valid BCA we now want to show the reverse, namely that every valid BCA has a corresponding s-t path. This way we can show that if there is a valid BCA, we can find it in the realization graph.

Lemma 6.10. *Let $E \subseteq E_P$ be an input edge set, and $H = (V, A)$ be the corresponding realization graph. Let $\sigma : P \to \mathbb{C}_2$ be a valid BCA. Then there exists an s-t path Q in H such that the corresponding BCA $\sigma_Q : P \to \mathbb{C}_2$ equals σ (possibly after swapping the colors in σ).*

Proof. First, assume that σ assigns color 1 to the first block, i.e., the same color assigned by all BCAs σ_Q for any corresponding to an s-t path Q. That is, if $\sigma(p_1) = \{2\}$ then swap colors 1 and 2 in σ. Let $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the special edges. We now construct an s-t path Q by selecting its nodes of H. Those are obviously s, t, and $\overleftrightarrow{u}_i$ for all small gaps $u_i \in \mathscr{S}_E \cap \mathscr{G}_E^-$. In addition, for all local maxima $\{p_a, p_{a+1}\} = u_i \in \mathscr{S}_E \cap \mathscr{M}_E$ select either $\overleftarrow{u}_i$ if $\sigma(p_a) = \{1, 2\}$ or $\overrightarrow{u}_i$ if $\sigma(p_{a+1}) = \{1, 2\}$. Since σ is a BCA, we know that exactly one endpoint of each local maximum has two colors and thus exactly one node for each local maximum is selected. As a result we have selected exactly one node for ↪

each $u_i \in \mathcal{S}_E$, let those nodes be called $\tilde{u}_i \in \{\bar{u}_i, \bar{u}_i, \vec{u}_i\}$ for $u_i \in \mathcal{S}_E$. Then we define the s-t path as $Q = (s, \tilde{u}_1, \dots, \tilde{u}_m, t)$.

We now need to show that Q is an actual path in H. From H's construction it is clear that there is an arc from s to $\tilde{u}_1$ and an arc from $\tilde{u}_m$ to t. It remains to show that for all $1 \le i < m$ the arc $(\tilde{u}_i, \tilde{u}_{i+1})$ exists. Let $\{p_a, p_{a+1}\} = u_i$ and $\{p_b, p_{b+1}\} = u_{i+1}$ be the corresponding special edges. In the construction of H all arcs are added that do not create a conflict with Lemma 6.7. Thus, there can be no interference with the restricted regions corresponding to $\tilde{u}_i$ and $\tilde{u}_{i+1}$, and as a result, the arc $(\tilde{u}_i, \tilde{u}_{i+1})$ exists in H. In conclusion, Q is an actual s-t path in H, and we have the corresponding BCA σ_Q.

The last step is now to show that $\sigma = \sigma_Q$. Both color assignments are BCAs and start with color 1 for B_0 which also means that the colors for all other blocks are fixed and thus the same in σ and σ_Q. The only possible difference is the choice which of the two endpoints of each local maximum should have two colors. Since we select $\overleftarrow{u}_i$ for Q if u_i's left endpoint has two colors in σ the result is that the same is true in σ_Q. Similarly, we select $\vec{u}_i$ for Q if u_i's right endpoint has two colors in σ and as a result the same is true in σ_Q. Thus, we can conclude that $\sigma = \sigma_Q$. □

6.4.3 The Algorithm

With the previous sections we have already done most of the work for the actual algorithm that finds a color assignment for a given input. We can now describe this algorithm very concisely and then show its correctness as well as its linear running time.

The input to the algorithm is an input edge set $E \subseteq E_P$ where $P \subseteq \mathbb{R}$ is an input point. The goal is to find a valid color assignment $\sigma : P \to \mathbb{C}_2$ with two colors. As the first step in the algorithm we construct the realization graph H for E. We then look for an s-t path Q in H. If no such path exists, we return "E cannot be colored with two colors". Otherwise, we return σ_Q, the corresponding color assignment.

As we can see, the top level algorithm is rather short. We first show that it actually runs in linear time.

Lemma 6.11. *Given an input edge set for an input point set with n points, the algorithm in Section 6.4.3 runs in $O(n)$ time.*

Proof. Let k be the number of local maxima in E, ℓ be the number of small gaps, and $m = k + \ell$ be their sum. From Observation 6.3 we know that $m < \lfloor n/2 \rfloor$.

In the first step we construct the realization graph. H contains $2 + 2k + \ell \le n + 2$ nodes and at most $4(m-1) + 4 = 4m \le 2n$ edges. For each potential edge we need to check at most two points whether they are inside a restricted region, which can be done in constant time per check. Thus, the construction of H takes $O(n)$ time. Finding ↪

the s-t path Q can be done by breadth-first or depth-first search which both take $O(n)$ time. Finally, constructing σ_Q is also done in linear time: We have one pointer to the points and one to the path Q and both are only advanced from left to right. In the simplest case a point's color is defined as the previous point's color. Only when the point is an endpoint of the special edge corresponding to the current node in Q we assign two colors or switch colors. $\square$

We also need to show that the algorithm is correct, that is, the returned color assignments result in the correct CNNG and it always finds a color assignment if one exists.

Lemma 6.12. *The algorithm in Section 6.4.3 is correct. That is, given an input edge set $E \subseteq E_P$, the algorithm returns a valid BCA $\sigma : P \to \mathbb{C}_2$ if and only if a valid color assignment with two colors exists.*

Proof. Assume that a valid color assignment σ' with two colors exists. We can transform it to a valid BCA $\tilde{\sigma} : P \to \mathbb{C}_2$ according to Lemma 6.5. Furthermore, according to Lemma 6.10, a valid BCA implies the existence of an s-t path Q in the realization graph H for E such that the corresponding BCA σ_Q equals $\tilde{\sigma}$ (possibly after swapping the colors in $\tilde{\sigma}$). Thus, σ_Q is valid which concludes the first direction of the equivalence. For the other direction we already know from Lemma 6.9 that if an s-t path exists in H that the corresponding BCA σ_Q is valid. Thus, the algorithm is correct. $\square$

We can now conclude this section by combining the previous results plus the result in Lemma 6.2 for one color:

Theorem 6.1. *Let $E \subseteq E_P$ be an input edge set for the input edge set $P \subseteq \mathbb{R}$. We can solve the* 1D-CNNG-Gaps *problem for P and E in linear time in the size of P (and thus E) for both $\hat{c} = 1$ and $\hat{c} = 2$.*

Proof. For $\hat{c} = 1$ we have shown in Lemma 6.2 that we can solve the problem in linear time. For $\hat{c} = 2$ we have the algorithm in the beginning of Section 6.4.3 which runs in linear time according to Lemma 6.11 and is correct as shown in Lemma 6.12. $\square$

Remember that solving the 1D-CNNG-Gaps problem in linear time for $\hat{c} \in \{1, 2\}$ means that we can also solve the 1D-CNNG problem in linear time for up to two colors. The property we observed in Section 6.4.1, which is that we only need to check neighboring special edges for conflicts with wrongly colored points, is restricted to the case with two colors. This means that the presented algorithm cannot be easily extended to more colors. In Chapter 8 we will present a dynamic program to solve the problem with arbitrarily many colors but only after looking at the bounds on the number of colors needed in Chapter 7.

Chapter 7
Different Bounds on the Number of Colors

After solving the problem for up to two colors, the natural next goal is to try to solve it for an arbitrary number of colors. However, we defer this to Chapter 8. We first want to look at the worst case behavior of our problems, more specifically at the number of colors needed to solve a given instance in the worst case. We look at the number of colors with respect to the various parameters of the input. At first thought the number of points n in the input point set seems to be the most relevant parameter, and we will also describe all our bounds with respect to n. However, as we could already see before, the most relevant part of each input is the set of special edges. Thus, the number of local maxima k, the number of small gaps ℓ, and the total number of special edges m will be the first parameter we are interested in.

In Section 7.1 we first show that every instance for the 1D-CNNG problem (the one without gaps) can be solved with at most a logarithmic number of colors with respect to k. We also present a construction that always needs that many colors. We will also note that this generalizes to the 1D-CNNG-Gaps problem as long as all gaps are big gaps, i.e., $\ell = 0$. For the general 1D-CNNG-Gaps problem we will see in Section 7.2 that we are only able to give a linear upper bound on the number of colors needed. In fact, we can also construct instances for the 1D-CNNG-Gaps problem that cannot be solved with fewer than a linear number of colors. These instances will rely only on small gaps and will include no local maxima, i.e., $k = 0$. In Section 7.3 we then combine the previous results into a bound that depends logarithmically on the number of local maxima and linearly on the number of small gaps. We also provide a construction to show that this bound is tight.

7.1 A Logarithmic Bound for Inputs Without Small Gaps

We start with the bounds by looking at the 1D-CNNG-Opt problem. For this problem, we want to find an upper bound on the number of colors needed to find a valid color assignment. In this situation we do not have any gaps which means that the only special edges we need to take care of are local maxima.

7.1.1 The Upper Bound

The only special edges we have are local maxima. Thus, for each local maximum we have to choose which endpoint should get two colors. On the other hand, for each local maximum we can base this decision on the number of different colors contained in each of its restricted regions. If we choose the endpoint such that the respective restricted region contains as few colors as possible, we will only add a new color if the restricted regions on both sides need the same number of colors. This is the idea for

Lemma 7.1. *Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ with $n \geq 2$ be an input point set and its neighbor edges E_P be the corresponding input edge set with k local maxima. Then there exists a valid BCA $\sigma : P \to \mathbb{C}_{\hat{c}}$ with $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor \leq \lfloor \log n \rfloor$.*

Proof. We show this by induction on the number of local maxima k. We first note that, since we don't have any gaps, the only special edges that can force us to use more colors are other local maxima.

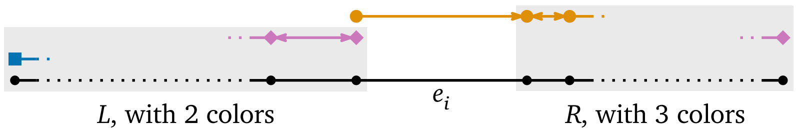
Induction base $k = 0$. If we have no local maxima we know from Lemma 6.1 that the NNG of P contains all edges from E_P and thus one color is sufficient.

Induction step $k-1 \to k$. Let $e_i = \{p_i, p_{i+1}\} \in \mathscr{M}(E_P)$ be a largest local maximum, i.e., $\|e_i\| \geq \|e\|$ for all $e \in \mathscr{M}(E_P)$. We partition P into two sets $L = \{p \in P \mid p \leq p_i\}$ and $R = \{p \in P \mid p \geq p_{i+1}\}$. Let $E_L \subsetneq E_P$ and $E_R \subsetneq E_P$ be those edges from E_P that have both endpoints in L and R, respectively. Then we know that $E_P = E_L \uplus E_R \uplus \{e_i\}$. Because e_i is a largest local maximum we know that for any local maximum e in L the restricted regions of e do not contain any points from R: Any restricted region is at most as wide as e_i itself and it starts to the left of p_i. By symmetry no restricted region of a local maximum in R contains any points from L. As a result, we can combine any valid color assignment for L with any valid color assignment for R to obtain $E_P \setminus \{e_i\}$.

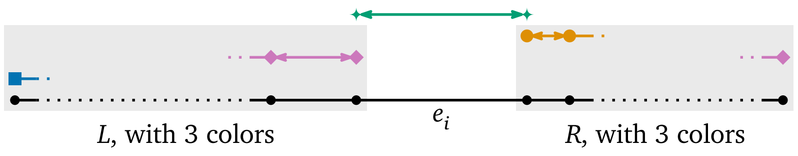
Let k_L and k_R be the number of local maxima in E_L and E_R, respectively. We know that $k = k_L + k_R + 1$ and thus $k_L < k$ and $k_R < k$ which means that we can apply the induction hypothesis to L and R. Hence, we know that there is a valid BCA σ_L for L with at most $\hat{c}_L = 1 + \lfloor \log(k_L + 1) \rfloor$ colors and a valid BCA σ_R for R with at most $\hat{c}_R = 1 + \lfloor \log(k_R + 1) \rfloor$ colors. Assume, without loss of generality, that $\hat{c}_L \leq \hat{c}_R$.

If $\hat{c}_L < \hat{c}_R$ then there is a color c used to color points in R that is not used for any point in L. As observed before, just combining σ_L and σ_R to σ would result in a valid BCA for $E_P \setminus \{e_i\}$, i.e., $\mathscr{N}(P, \sigma) = E_P \setminus \{e_i\}$. In order to also include e_i we rename the colors in σ_R such that the leftmost block (with respect to R) is given color c which also means that $\sigma_R(p_{i+1}) = \{c\}$. We construct σ by using σ_L for all points in L and σ_R for all points in R. We then make a small change to σ by additionally assigning c to p_i. See Figure 7.1a for an example of this situation. Since no other point in L has this color, $p_i \overset{c}{\underset{\sigma}{\circ\!\!\rightarrow}} p_{i+1}$. On the other hand $p_{i+1} \overset{c}{\underset{\sigma}{\circ\!\!\rightarrow}} p_{i+2}$ since e_i is a local maximum. No other point in R will connect to p_i in color c since p_{i+1} is closer. We then have obtained a valid

↪



(a) L needs fewer colors than R. We thus assign a color not used for L to the left endpoint of e_i.



(b) L and R need the same number of colors. We thus assign a new color to the endpoints of e_i.

Figure 7.1: The two possible situations from the induction step in the proof of Lemma 7.1.

BCA which uses $\hat{c} = \hat{c}_R$ colors. By induction $\hat{c}_R = 1 + \lfloor \log(k_R + 1) \rfloor \leq 1 + \lfloor \log(k + 1) \rfloor$ which proves the upper bound for $\hat{c}$.

If $\hat{c}_L = \hat{c}_R$, then $\tilde{c} = \hat{c}_R + 1$ is the smallest color not used by either σ_L or σ_R. We then, again, construct σ by using σ_L for all points in L and σ_R for all points in R. As an addition we give the new color $\tilde{c}$ to both p_i and p_{i+1}. See Figure 7.1b for an example of this situation. No other point has color $\tilde{c}$ and thus $p_i \overset{\tilde{c}}{\underset{\sigma}{\circ\!\!\rightarrow}} p_{i+1}$ and $p_{i+1} \overset{\tilde{c}}{\underset{\sigma}{\circ\!\!\rightarrow}} p_i$. The other colors do not interfere with each other which means that σ is a valid color assignment with $\hat{c} = \hat{c}_R + 1$ colors. It is not a BCA since both endpoints of e_i have two colors. However, using Lemma 6.5 we can transform σ into a BCA.

We now need to take a closer look at $\hat{c}_L$ and $\hat{c}_R$ to show that $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor$. In order for $\hat{c}_L = \hat{c}_R$ to hold it must be that $k_L = 2^{\hat{c}_L - 1} + s_L - 1$ and $k_R = 2^{\hat{c}_L - 1} + s_R - 1$ with $0 \leq s_L, s_R < 2^{\hat{c}_L - 1}$. We then have that

$$
\begin{aligned}
1 + \lfloor \log(k+1) \rfloor &= 1 + \lfloor \log(k_L + k_R + 1 + 1) \rfloor \\
&= 1 + \left\lfloor \log\left(2^{\hat{c}_L - 1} + s_L + 2^{\hat{c}_L - 1} + s_R\right) \right\rfloor \\
&= 1 + \left\lfloor \log\left(2^{\hat{c}_L} + s_L + s_R\right) \right\rfloor \\
&= 1 + \hat{c}_L,
\end{aligned}
$$

where the last step is valid since $0 \leq s_L + s_R < 2^{\hat{c}_L}$. Then, since $\hat{c} = \hat{c}_L + 1$ we know that $\hat{c} \leq \hat{c}_L + 1 \leq 1 + \lfloor \log(k+1) \rfloor$, as claimed.

It now only remains to show that $1 + \lfloor \log(k+1) \rfloor \leq \lfloor \log n \rfloor$. For this, observe that $1 + \lfloor \log(k+1) \rfloor = \lfloor \log(2k+2) \rfloor$ which means that we only need to show that $2k + 2 \leq n$. From Observation 6.3 we know that $2m + 2 \leq n$ where m is the number of special edges. Since $k \leq m$, or in this case $k = m$, the result follows immediately. $\square$

The proof can actually be applied to input edge sets with gaps, as long as they are all big gaps. For the induction base we know that we can color a point set with only big gaps (and no local maxima) with just one color, as the NNG automatically excludes the local maxima in E_P (compare Lemma 6.1). In the induction step we only use that all special edges are local maxima but not that there are no gaps in the input edge set. Thus, we can conclude the following

Corollary 7.1. *Let $E \subseteq E_P$ be an input edge set with k local maxima and no small gaps for an input point set $P \subseteq \mathbb{R}$ with n points. Then there exists a valid BCA $\sigma : P \to \mathbb{C}_{\hat{c}}$ with $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor \leq \lfloor \log n \rfloor$.* □

7.1.2 The Matching Lower Bound

We now want to find a construction for a point set that is guaranteed to need at least a logarithmic number of colors for any valid BCA. To this end, the question we can ask ourselves is: How can we force a local maximum to need to use an additional color, and how can we do this with the lowest number of special edges possible? The answer is a construction where we add a local maximum such that both its left and right restricted region overlaps as many other local maxima as possible. If both restricted regions contain the same number of colors we will need to add a new color for the new local maximum. This what we did in the proof of Lemma 7.1 where $\hat{c}_L = \hat{c}_R$. The following recursively constructed point set is exactly what we need.

Definition 7.1. For every number $a \in \mathbb{N}^+$ we define a point set $\mathsf{P}(a)$ recursively as

$$\begin{aligned} \mathsf{P}(1) &= \{0, 1\}, \\ \mathsf{P}(a) &= \underbrace{\mathsf{P}(a-1)}_{L(a)} \cup \underbrace{\{p + 1 + 2\max \mathsf{P}(a-1) \mid p \in \mathsf{P}(a-1)\}}_{R(a)}. \end{aligned}$$

See Figure 7.2 for a depiction of the point set and its recursive structure for $\mathsf{P}(3)$. There we also have an indicator to see that the distance between the two parts $L(a)$ and $R(a)$ is one unit larger than the width of each individual part. To make $\mathsf{P}(a)$ be a possible input for the 1D-CNNG problem, it must be an input point set. This can, of course, be achieved by a slight random perturbation of all points, but we can easily show that this is not necessary:

Lemma 7.2. *For every number $a \in \mathbb{N}^+$ the point set $\mathsf{P}(a)$ is an input point set.*

Proof. We need to show that no point has two other points at the same distance. We do this by induction on a.

Induction base $a = 1$. Here we have only two points and the claim trivially holds. ↪

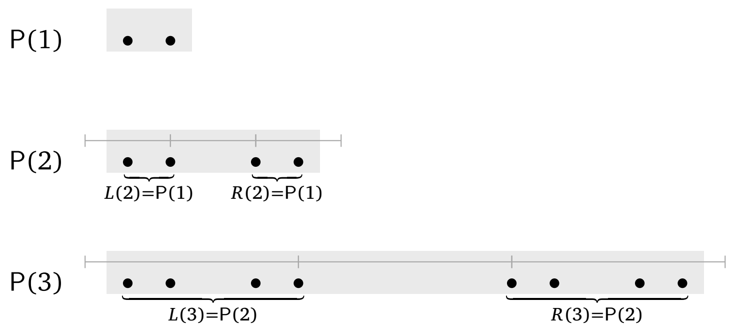


Figure 7.2: The progressive construction of the lower bound $\mathsf{P}(3)$.

Induction step $a-1 \to a$. We know by induction that $L(a)$ and $R(a)$ are input point sets. Thus, we only need to show that combining the two sets does not violate the property. We show that no point in $L(a)$ has a point in $R(a)$ that is at the same distance as another point from $L(a)$. Due to symmetry the argument then extends to $R(a)$ as well. Look at the rightmost point in $L(a)$ which is at position $p_l = \max \mathsf{P}(a-1)$. On the other hand, the leftmost point in $R(a)$ is at position $p_r = 2\max\mathsf{P}(a-1)+1$. Then $\operatorname{dist}(p_l, p_r) = \max\mathsf{P}(a-1)+1$ and the point to the left of p_l with same distance as p_r is point -1. Since the leftmost point in $L(a)$ and thus $\mathsf{P}(a)$ is 0 we see that no such point exists. For all other points $p \in L(a)$ and $q \in R(a)$ the distance between them is larger than $\max\mathsf{P}(a-1)+1$ and thus the point to the left of p with same distance as p is even further to the left than -1. □

We can now show that the 1D-CNNG problem with $\mathsf{P}(a)$ as its input we cannot find a valid color assignment with fewer than a colors. Have a look at Figure 7.3 to see an example color assignment and its BCA for $\mathsf{P}(3)$. In the upper of the two color assignments you can see the structure that both restricted regions of the local maximum contain all endpoints of all shorter local maxima to their left and right. This ultimately forces the use of a new color for each additional level of the construction, which we prove in

Lemma 7.3. *For every number $a \in \mathbb{N}^+$ we need at least a colors for any valid color assignment that solves the* 1D-CNNG *problem for the point set* $\mathsf{P}(a)$.

Proof. We show the claim by induction over a and remember that we want to obtain the input edge set $E_{\mathsf{P}(a)}$.

Induction base $a = 1$. Here we have $\mathsf{P}(1) = \{0,1\}$ and thus $E_{\mathsf{P}(1)} = \{\{0,1\}\}$. To obtain an edge we always need at least one color.

Induction step $a-1 \to a$. We use the induction assumption to know that every valid color assignment for $\mathsf{P}(a-1)$ needs at least $a-1$ colors. Let $p_l = \max\mathsf{P}(a-1) =$ ↪

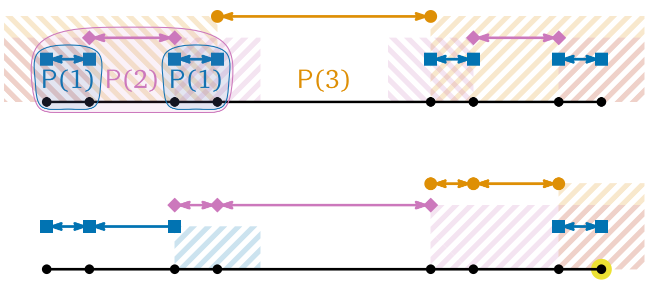


Figure 7.3: Top: A possible color assignment for P(3) where for every recursion step a new color is introduced to connect the local maximum in the middle. Bottom: The result of converting the color assignment to a BCA. We see that the marked point is inside two overlapping restricted regions of different colors which requires the third color here.

$\max L(a)$ be the rightmost point of $L(a)$ and let $p_r = 2p_l + 1 = \min R(a)$ be the leftmost point of $R(a)$. Then $e = \{p_l, p_r\}$ is the local maximum in the middle of $\mathsf{P}(a)$. Due to the construction we have that $L(a) \subseteq \overleftarrow{R}_E(e)$ and $R(a) \subseteq \overrightarrow{R}_E(e)$. Then there are points with $a-1$ colors in both of e's restricted regions. Thus, giving one of p_l and p_r two colors means that in order to connect to the other endpoint of e we must introduce a new color. As a result, we need at least a colors in any valid color assignment. □

We can easily see that $\mathsf{P}(a)$ has 2^a points, since for $a = 1$ we have two points and in each recursive step we double the number of points. At the same time, looking at the construction, we see that out of the $2^a - 1$ edges in $E_{\mathsf{P}(a)}$ every other edge is a local maximum which results in $2^{a-1} - 1$ local maxima. This can also be seen easily by induction where for $a = 1$ we have only one edge and thus no local maximum. In each recursion step we double the number of local maxima of the next smaller input and add one in the middle, resulting in $2\left(2^{(a-1)-1} - 1\right) + 1 = 2^{a-1} - 1$ local maxima. This means that for every power of two we already have a point set that needs a logarithmic number of colors. We will now see that for numbers that are not powers of two we can just remove some points from the point set constructed for the next larger power of two.

Lemma 7.4. *For every $n \in \mathbb{N}$ with $n \geq 2$ there is an input point set $P \subseteq \mathbb{R}$ with n points and $k = \lfloor n/2 \rfloor - 1$ local maxima in E_P such that there is no valid color assignment with fewer than $1 + \lfloor \log(k+1) \rfloor = \lfloor \log n \rfloor$ colors for E_P.*

Proof. Let $a = \lfloor \log n \rfloor = 1 + \lfloor \log(k+1) \rfloor$. Then we know from Lemma 7.3 that $\mathsf{P}(a)$ is a point set which needs a colors for the 1D-CNNG problem. Additionally, we saw that $\mathsf{P}(a)$ contains $k' = 2^{a-1} - 1$ local maxima and $n' = 2^a = 2k' + 2$ points. Then

↪

$\mathsf{P}(a)$ is missing $k - k'$ local maxima and $n - n'$ points which equals $2(k - k')$ if n is even and $2(k - k') + 1$ otherwise.

The easiest solution is to just take $\mathsf{P}(a + 1)$ which consists of two copies of $\mathsf{P}(a)$ and remove points from the right until the correct number of local maxima and points is reached. Thus, $P = \{p \in \mathsf{P}(a + 1) \mid p \leq p_n\}$ where p_n is the nth point in $\mathsf{P}(a + 1)$ from the left. Then P fulfills all the desired properties, firstly that it has n points. Since, as observed before, every other edge in $E_{\mathsf{P}(a+1)}$ is a local maximum the same holds true for E_P. Thus, E_P contains $k = \lfloor n/2 \rfloor - 1$ local maxima. Additionally, P is an input point set since $\mathsf{P}(a + 1)$ is an input point set and removing points from an input point set obviously does not destroy that property. Furthermore, P contains $\mathsf{P}(a)$ with the other points all to the side of it. Every special edge e from $E_{\mathsf{P}(a)}$ is also a special edge in E_P. All endpoints of other special edges that are contained in e's restricted regions with respect to $E_{\mathsf{P}(a)}$ are also contained by them with respect to E_P. Thus, the number of colors needed to color P is at least as large as the number of colors needed to color $\mathsf{P}(a)$ which is a, exactly as claimed.

We want to briefly look at the size of the coordinates in P, to see that they are not too large to represent P with a polynomial number of bits. For this we look at the rightmost point of $\mathsf{P}(a)$ which we call $r(a)$. Then $r(1) = 1$ and $r(a) = 3r(a - 1) + 1$ which can be described as $r(a) = \sum_{i=0}^{a-1} 3^i = \frac{3^a - 1}{2} < 3^a$. Then we know that the rightmost point in P is smaller than $r(a + 1) < 3^{a+1} = 3^{\lfloor \log n \rfloor + 1} \leq 3^{1 + \log n} \approx 17.11n$. As a result, every point in P can be represented by $O(\log n)$ bits and thus P has size $O(n \log n)$. $\square$

This lower bound is a lower bound for the 1D-CNNG problem which then automatically is a lower bound for the 1D-CNNG-Gaps problem. In summary, we have now seen that $1 + \lfloor \log(k + 1) \rfloor \leq \lfloor \log n \rfloor$ is a tight bound for the number of colors needed to solve the 1D-CNNG problem and the more general 1D-CNNG-Gaps problem without small gaps. In the next section we will then see what happens if we allow small gaps in the input edge set.

7.2 A Linear Bound for Inputs With Small Gaps

We now turn our focus towards the general 1D-CNNG-Gaps problem where we also allow small gaps. Even though we have seen with Corollary 7.1 that we can apply the result of the logarithmic upper bound to a restricted subset of 1D-CNNG-Gaps instances, we want to obtain bounds for all possible inputs.

7.2.1 The Upper Bound

The first obvious linear upper bound on the number of colors needed for a valid color assignment is that we can always find a color assignment with one color for each edge in the input edge set.

Lemma 7.5. *Given an input edge set $E \subseteq E_P$ for an input point set $P \subseteq \mathbb{R}$ with $n \geq 2$ points, there exists a valid color assignment $\sigma : P \to \mathbb{C}_{\hat{c}}$ for E such that $\hat{c} = |E| \leq n - 1$.*

Proof. We assign each edge $e_i \in E$ its own color c_i and both endpoints are given this color in σ. This means that a point that has two incident edges obtains the two colors of both edges, i.e., $\sigma(p_i) = \{c_{i-1}, c_i\}$. Since for every edge $e_i = \{p_i, p_{i+1}\} \in E$ we have $p_i \overset{c_i}{\multimap} p_{i+1}$ and $p_{i+1} \overset{c_i}{\multimap} p_i$, it follows directly that σ is valid. □

We can easily see that it is unreasonable to use one color for each edge: We know that for every valid color assignment we can find a valid BCA, and we know that a BCA needs at most as many colors as there are blocks in the input edge set. As a result, we can improve out upper bound slightly:

Lemma 7.6. *Given an input edge set $E \subseteq E_P$ with m special edges for an input point set $P \subseteq \mathbb{R}$ with $n \geq 2$ points, there exists a valid BCA $\sigma_P : P \to \mathbb{C}_{\hat{c}}$ for E such that $\hat{c} \leq m + 1 \leq \lfloor n/2 \rfloor$.*

Proof. We take the color assignment obtained in Lemma 7.5 and convert it into a BCA $\sigma : P \to \mathbb{C}_{\hat{c}}$ using Lemma 6.5. Then, we have at most as many colors $\hat{c}$ as we have blocks in E which means that $\hat{c} \leq m + 1$. From Observation 6.3 we know that $m \leq \lfloor n/2 \rfloor - 1$ which concludes the proof. □

Even though an upper bound of $m + 1$ or $\lfloor n/2 \rfloor$ seems to be disappointing at first, we will see in the next section that there is a linear lower bound matching our upper bound.

7.2.2 The Matching Lower Bound

We now present a construction for arbitrarily large input edge sets that will need $m = \lfloor n/2 \rfloor$ colors in any valid color assignment where m is the number of special edges. The construction is recursively defined and for each recursion step we add two points with an edge between them, effectively creating one small gap and one edge.

Definition 7.2. For every number $a \in \mathbb{N}^+$ we define a point set $\mathsf{Q}(a)$ and a corresponding edge set $\mathsf{E}(a) \subseteq E_{\mathsf{Q}(a)}$ recursively as follows:

$$\begin{aligned} \mathsf{Q}(1) &= \{0.5, 2\}, \\ \mathsf{E}(1) &= \{\{0.5, 2\}\}, \\ \mathsf{Q}(a) &= \mathsf{Q}(a-1) \cup \{p+1, 2p+1\}, \text{ and} \\ \mathsf{E}(a) &= \mathsf{E}(a-1) \cup \{\{p+1, 2p+1\}\} \\ &\quad \text{with } p = \max \mathsf{Q}(a-1). \end{aligned}$$

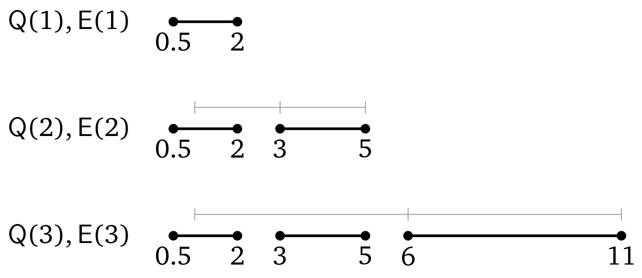


Figure 7.4: The progressive construction of Q(3) and E(3). The gray indicator already shows that the left restricted region of the rightmost small gap contains all points to its left except 0.5.

See Figure 7.4 for a depiction of the construction up to $a = 3$. We first make some observations about those sets to see that they are an input point set and an input edge set. We also observe that their positions can be easily defined non-recursively.

Lemma 7.7. *For every $a \in \mathbb{N}^+$ we have that $\mathsf{Q}(a)$ contains $2a$ points and $a - 1$ small gaps. Furthermore,*

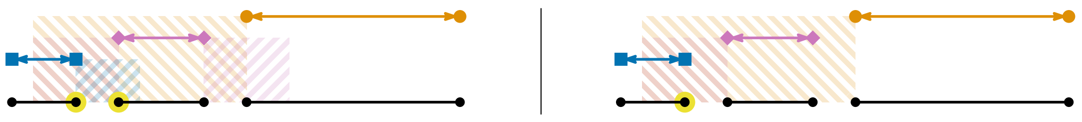
$$\mathsf{Q}(a) = \{0.5, 2\} \cup \bigcup_{2 \le i \le a} \left\{3 \cdot 2^{i-2}, 3 \cdot 2^{i-1} - 1\right\}.$$

In addition, $\mathsf{Q}(a)$ is an input point set and $\mathsf{E}(a)$ is an input edge set.

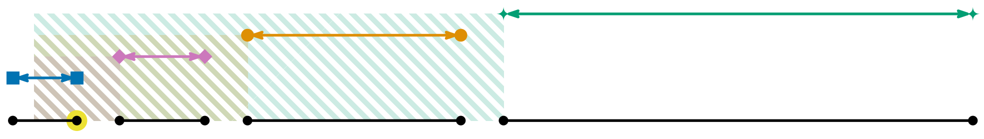
Proof. The first claim is obvious since each recursion step adds 2 points to $\mathsf{Q}(a)$ and for $a = 1$ we have 2 points. For the second claim we observe that $\mathsf{E}(1)$ does not contain any gaps. However, in every recursion step we add two points and just one edge which means that we add one gap. This gap has length 1 while the edges on both sides have length at least 1.5, making it a small gap.

For the third claim we observe that the rightmost point $r(a)$ in $\mathsf{Q}(a)$ is defined as $r(1) = 2$ and $r(a) = 2r(a-1) + 1$ which can be written in closed form as $r(a) = 2^a + 2^{a-1} - 1 = 3 \cdot 2^{a-1} - 1$. In addition, we call the left of the two points added to $\mathsf{Q}(a)$ in the recursion $l(a)$ and see that $l(a) = r(a-1) + 1 = 3 \cdot 2^{a-2} - 1 + 1 = 3 \cdot 2^{a-2}$. In each step i of the recursion we add $l(i) = 3 \cdot 2^{i-2}$ and $r(i) = 3 \cdot 2^{i-1} - 1$, just as claimed.

To ensure that $\mathsf{Q}(a)$ is an input point set we need to check that no point has two points with equal distance, one to the left, one to the right. This is obviously true for $a = 1$. The added point $r(a)$ does not have any point to its right, so the condition is trivially fulfilled. For $l(a)$ we know that the only point to its right has distance $r(a-1)$ which means that there must not be a point at position $l(a) - r(a-1) = 1$ which there isn't. We also need to check that no other point $p \in \mathsf{Q}(a-1)$ has a point p' to its left such that the distance from p to p' is the same as the distance from p to $l(a)$ or ↪



(a) On the left we see the all restricted regions while on the right we only see the interesting left restricted regions.



(b) We only show the left restricted regions here.

Figure 7.5: The only way to color $\mathsf{Q}(i)$ to obtain $\mathsf{E}(i)$ for $i = 3$ in (a) and for $i = 4$ in (b). The marked points are those inside the most overlapping restricted regions of different colors. These points are thus witnesses for the need of three or four colors.

to $r(a)$. We first check $r(a-1)$: The distance between $r(a-1)$ and $l(a)$ is 1 and by construction the distance between $r(a-1)$ and $l(a-1)$ is $r(a-2) \geq 2$ and thus there is no point at $r(a-1)-1$. The distance between $r(a-1)$ and $r(a)$ is $r(a-1)+1$ and there is no point at -1. We now check $l(a-1)$ and can easily see that $l(a) = 2l(a-1)$ from the already shown second claim. Thus, there must not be point at position 0 which there isn't. Now, the distance between $l(a-1)$ and $r(a)$ is even larger and thus $l(a-1) - (r(a) - l(a-1)) < 0$, but there are no negative points in $\mathsf{Q}(a)$ for any $a \in \mathbb{N}^+$. Similarly, the distance between any point $p \in \mathsf{Q}(a-2)$ and $l(a)$ or even $r(a)$ is much larger than p and thus the position at which no point is allowed is negative. This shows that $\mathsf{Q}(a)$ is indeed an input point set.

It is obvious that $\mathsf{E}(a)$ is an input point set as every pair of points added in each recursion step has an edge added between those two points; thus every point has exactly one incident edge. □

Using these previous results we can now show that this input point and input edge set is one that cannot be colored with fewer than a linear number of colors. See Figure 7.5 for the structurally only way to color $\mathsf{Q}(i)$ to obtain $\mathsf{E}(i)$ for $i \in \{3,4\}$.

Lemma 7.8. *For every natural number $n \in \mathbb{N}$ with $n \geq 2$ there is an input point set $P \subseteq \mathbb{R}$ with n points and a corresponding input edge set $E \subseteq E_P$ with $m = \lfloor n/2 \rfloor - 1$ special edges such that $m+1 = \lfloor n/2 \rfloor$ is the optimal number of colors for the* 1D-CNNG-Gaps *problem.*

Proof. Let $a = \lfloor n/2 \rfloor$. If n is even, let $P = \mathsf{Q}(a)$ and $E = \mathsf{E}(a)$ be the input point and input edge set. If n is odd, let $P = \{0.1\} \cup \mathsf{Q}(a)$ and $E = \{\{0.1, 0.5\}\} \cup \mathsf{E}(a)$. To see ↪

that P is still an input point set in the latter case, we observe that all points except 0.1 and 0.5 have integer coordinates; thus the distance from any point to 0.1 is non-integer and to any other point (except 0.5) is integer. For 0.1 and 0.5 we can easily see that there is no other point at 0.9. As a result, it is clear that E is also still an input edges set since the additional point has an incident edge to its immediate neighbor.

Now P contains exactly $m = a - 1$ small gaps according to Lemma 7.7. We show by induction that P cannot be colored with fewer than a points. For $a = 1$ (that is $n \in \{2, 3\}$) we clearly need at least one color. For larger a (that is $n \in \{2a, 2a + 1\}$) we use the induction assumption to know that we need $a - 1$ colors to color $\mathsf{Q}(a - 1)$ to obtain $\mathsf{E}(a - 1)$. The two additional points in $\mathsf{Q}(a)$, call them $l(a)$ and $r(a)$ again as in Lemma 7.7, need to have the same color. However, they cannot have the same color as any point inside the left restricted region of the small gap between $r(a - 1)$ and $l(a)$, given by $\overleftarrow{R}_E(\{r(a-1), l(a)\})$, as otherwise $l(a)$ would not connect to $r(a)$. We can see from its definition that

$$\begin{aligned}
\overleftarrow{R}_E(\{r(a-1), l(a)\}) &= \overleftarrow{R}(l(a), r(a)) \\
&= [2l(a) - r(a), l(a)) \\
&= \left[2 \cdot 3 \cdot 2^{a-2} - \left(3 \cdot 2^{a-1} - 1\right), l(a)\right) \\
&= \left[3 \cdot 2^{a-1} - 3 \cdot 2^{a-1} + 1, l(a)\right) \\
&= [1, l(a)).
\end{aligned}$$

Since all points in P except 0.5 (and 0.1 if n is odd) are larger than 1 they are all included in $\overleftarrow{R}_E(\{r(a-1), l(a)\})$. This includes all endpoints of all small gaps in P. As a result, all $a - 1$ colors needed to color $\mathsf{Q}(a - 1)$ to obtain $\mathsf{E}(a - 1)$ are contained in $\overleftarrow{R}_E(\{r(a-1), l(a)\})$. Thus, we need an additional ath color for $l(a)$ and $r(a)$.

As in the proof for Lemma 7.4 we want to look at how many bits we need to represent P and E. It is clear that E can be represented as a set of integers of total size $|E| \log n$. The rightmost point $r(a)$ in P has coordinate $3 \cdot 2^{\lfloor n/2 \rfloor - 1} - 1$ which means that we need $\log\left(3 \cdot 2^{\lfloor n/2 \rfloor - 1} - 1\right) \leq \log 3 + \lfloor n/2 \rfloor - 1$ bits to represent it. As a result, we can conclude that P and E can be represented with $O\left(n^2\right)$ bits. □

In summary, we have now seen that $m + 1 \leq \lfloor n/2 \rfloor$ is a tight bound for the number of colors needed to solve the 1D-CNNG-Gaps problem.

7.3 A Mixed Bound Depending on Small Gaps and Local Maxima

Looking at the upper bound in Lemma 7.6 we see that for the 1D-CNNG-Gaps problem we need at most a linear number of colors, relative to the number of special edges. Restricting the special edges to local maxima, as seen in Lemma 7.1 and Corollary 7.1

we only have a logarithmic growth with respect to the number of local maxima. We have also seen that the linear lower bound from Section 7.2.2 uses a construction that solely relies on small gaps instead of local maxima. This leads us to believe that we should be able to introduce bounds that combine the previous results. They should be logarithmic in the number of local maxima while being linear in the number of small gaps.

7.3.1 The Upper Bound

Taking the mentioned ideas we can find an upper bound that perfectly combines the previous upper bound results, as we can see in

Lemma 7.9. *Let $E \subseteq E_P$ be an input edge set for an input point set P and let E have k local maxima, ℓ small gaps, and thus $m = k + \ell$ special edges. Then there exists a valid BCA with $\hat{c}$ colors such that $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor + \ell$.*

Proof. We show this by induction on both k and ℓ by first showing two base cases for $k = 0$ and $\ell = 0$ and then showing two induction steps, one for $k - 1 \to k$ and one for $\ell - 1 \to \ell$.

Induction base $k = 0$ or $\ell = 0$. For $k = 0$ the claim is that there exists a valid BCA for P with $\hat{c} \leq 1 + \lfloor \log 1 \rfloor + \ell = 1 + \ell$ colors. We have already shown in Lemma 7.6 that we need at most $m + 1$ colors for any input edge set where m is the number of special edges. In this case $m = \ell$ and the claim follows directly.

For $\ell = 0$ the claim is that there exist a valid BCA for P with $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor + \ell = 1 + \lfloor \log(k+1) \rfloor$ colors. We remember Corollary 7.1 which shows that if we don't have any small gaps but k local maxima then we can find a valid BCA with $1 + \lfloor \log(k+1) \rfloor$ colors, which concludes the induction base.

Induction step $k - 1 \to k$ or $\ell - 1 \to \ell$. For the induction step let $u \in \mathscr{S}_E$ be a special edge such that the length of its longest restricted region is maximized. Let $u = e_i = \{p_i, p_{i+1}\}$. We then partition P into two sets $L = \{p \in P \mid p \leq p_i\}$ and $R = \{p \in P \mid p \geq p_{i+1}\}$ and let E_L and E_R be the input edge sets restricted to points in L and R, respectively. Let k_L and ℓ_L be the number of local maxima and small gaps in L and k_R and ℓ_R the same in R. Since u is either a local maximum or small gap we know that either $k_L, k_R < k$ or $\ell_L, \ell_R < \ell$. This means that we can apply the induction hypothesis to L and R to see that there exist valid BCAs σ_L and σ_R for L and R with $\hat{c}_L \leq 1 + \lfloor \log(k_L + 1) \rfloor + \ell_L$ and $\hat{c}_R \leq 1 + \lfloor \log(k_R + 1) \rfloor + \ell_R$ colors, respectively.

Induction step $\ell - 1 \to \ell$ if u is a small gap. Here we have that $k_L + k_R = k$ and $\ell_L + \ell_R + 1 = \ell$. Assume, without loss of generality, that the larger of the two restricted regions of u is the left restricted region $\overleftarrow{R}_E(u)$. This means that e_{i+1}, the edge to the right of u, is responsible for the size of $\overleftarrow{R}_E(u)$.

↪

It then follows that in R there is no special edge that has p_{i+1} in its left restricted region: For this to happen the length of this restricted region would need to be greater than $\|e_{i+1}\|$. However, by assumption, no restricted region is larger than $\|e_{i+1}\|$. The result implies that there is no special edge in R that has a point from L in their left restricted region. With the same argument we can also see that no special edge in L has a point p_{i+2} in its right restricted region. Thus, the only point from R they may have in their right restricted regions is p_{i+1}.

We now have a look at the valid BCAs σ_L and σ_R. If we were to combine them we would have two problems: First, some endpoints of special edges in L with the same color as p_{i+1} may connect to p_{i+1}. In the other direction, p_{i+1} may not connect to p_{i+2} with its given color if some point in its left restricted region in L is given the same color as p_{i+1}. The solution to both problems is that we give a new color to the block to the right of u. Thus, p_{i+1} must connect to p_{i+2} and no point in L will connect to p_{i+1}.

As a result, we will need $\hat{c} = \max(\hat{c}_L, \hat{c}_R) + 1$ colors. It remains to show that this adheres to our claim. For this we look at the worst case, the largest $\hat{c}$ that can be achieved. Since the bound on the color numbers is monotone in both k and ℓ we can easily see that $\max(\hat{c}_L, \hat{c}_R)$ is maximized if one of the color numbers is as low as possible. Let this one be $\hat{c}_L$ without loss of generality. We thus set $k_L = \ell_L = 0$ which implies $k_R = k$ and $\ell_R = \ell - 1$. Then $\hat{c}_R \leq 1 + \lfloor \log(k+1) \rfloor + \ell - 1$ by induction. As a result we have that $\hat{c} = \hat{c}_R + 1 \leq 1 + \lfloor \log(k+1) \rfloor + \ell$, exactly as claimed.

Induction step $k-1 \to k$ if u is a local maximum. Here we have that $k_L + k_R + 1 = k$ and $\ell_L + \ell_R = \ell$. Since no restricted region of any other special edge is larger than $\|e_i\|$ we directly see that no special edge in L can contain any point from P or vice versa. Thus, we can already look at how to construct σ from σ_L and σ_R. We will now distinguish whether $\hat{c}_L$ and $\hat{c}_R$ are the same or different.

If $\hat{c}_L > \hat{c}_R$, we know that there is a color c in σ_L that is not used by σ_R. Rename the colors in σ_L such that it assigns c to the rightmost block in L, the points directly to the left of u. Then choose p_{i+1}, the right endpoint of u, for two colors. Since no special edge in L has a point from R in a restricted region, no point will connect to any of the points in R independent of their color. The point p_{i+1} will connect with the color given by σ_R to p_{i+2} because this is closer than any other point. And it will connect with c to p_i since no other point in R has this color. Thus, the number of colors needed is $\hat{c} = \hat{c}_L$ and we only need to show that this can be bounded as claimed. We know by induction that $\hat{c}_L \leq 1 + \lfloor \log(k_L + 1) \rfloor + \ell_L$ and since $k_L < k$ and $\ell_L \leq \ell$ we can immediately see that $\hat{c} \leq 1 + \lfloor \log(k+1) \rfloor + \ell$.

If $\hat{c}_L < \hat{c}_R$ we can, of course, apply the symmetric arguments which will also show that the bound holds.

If, on the other hand, $\hat{c}_L = \hat{c}_R$ we just combine σ_L and σ_R to form σ and add a new color to p_i and p_{i+1}, the same way we did in the proof of Lemma 7.1. Since the special edges in L and R don't have any points from the other set in their restricted regions,

↪

there is no problem in combining σ_L and σ_R. The new color for u also prevents any problem with points connecting to p_i and p_{i+1} or vice versa. Then, the number of colors needed is $\hat{c} = \hat{c}_L + 1 = \hat{c}_R + 1$. Even though σ is not a BCA, we can apply Lemma 6.5 to convert it into a BCA. We now have a closer look at $\hat{c}_L$ and $\hat{c}_R$. For this let

$$\begin{aligned}
a_L &= \lfloor \log(k_L + 1) \rfloor \quad \text{and thus} \quad & k_L &= 2^{a_L} + s_L - 1 \quad & \text{with } 0 \le s_L \le 2^{a_L},\\
a_R &= \lfloor \log(k_R + 1) \rfloor \quad \text{and thus} \quad & k_R &= 2^{a_R} + s_R - 1 \quad & \text{with } 0 \le s_R \le 2^{a_R}, \text{ and}\\
& & k &= 2^{a_L} + s_L + 2^{a_R} + s_R - 1. &
\end{aligned}$$

To prove that $\hat{c} \le 1 + \lfloor \log(k+1) \rfloor + \ell$ we look at whether a_L and a_R are the same or different:

If $a_L = a_R$ we can see that

$$\begin{aligned}
\lfloor \log(k+1) \rfloor &= \lfloor \log(2^{a_L} + s_L + 2^{a_R} + s_R) \rfloor\\
&= \left\lfloor \log\left(2^{a_L+1} + s_L + s_R\right) \right\rfloor\\
&= a_L + 1, \text{ since } 0 \le s_L + s_R < 2^{a_L+1},
\end{aligned}$$

which then immediately means that

$$\begin{aligned}
\hat{c} = \hat{c}_L + 1 &\le 1 + a_L + \ell_L + 1\\
&\le 1 + a_L + \ell + 1\\
&= 1 + \lfloor \log(k+1) \rfloor + \ell
\end{aligned}$$

which is exactly what we want to prove.

If $a_L > a_R$ then we also want to find out what $\lfloor \log(k+1) \rfloor$ is. However, we only know that $0 \le a_R < a_L$, but this is sufficient to find a lower bound for $\lfloor \log(k+1) \rfloor$. Since we have a monotone function in all four variables we can just use $s_L = s_R = a_R = 0$ for which then

$$\begin{aligned}
\lfloor \log(k+1) \rfloor &= \lfloor \log(2^{a_L} + s_L + 2^{a_R} + s_R) \rfloor\\
&\ge \lfloor \log(2^{a_L} + 0 + 0 + 0) \rfloor\\
&= a_L.
\end{aligned}$$

Since $a_L > a_R$ but also $a_L + \ell_L = a_R + \ell_R$ it must follow that $\ell_L < \ell_R \le \ell$ and thus

$$\begin{aligned}
\hat{c} = \hat{c}_L + 1 &\le 1 + a_L + \ell_L + 1\\
&\le 1 + a_L + \ell\\
&\le 1 + \lfloor \log(k+1) \rfloor + \ell
\end{aligned}$$

which we wanted to show. For $a_R > a_L$ the same holds due to the symmetry of all arguments. □

As already noted in the induction base, we can see that the upper bound that we have just shown is the combination of the two upper bounds shown before in Lemmas 7.1 and 7.6. If we don't have any small gaps and thus $\ell = 0$ we obtain the situation where only a logarithmic number of colors is needed. On the other hand, if we only have small gaps, that is $\ell = m$, we have the situation that in the worst case every additional small gap needs an additional color and we obtain the linear upper bound.

7.3.2 Running Time of the Construction

We can use the arguments used in the proof for Lemma 7.9 as an algorithm to find a (generally not optimal) solution for a given input. In the worst case, in each step of the recursion, the special edge with the largest restricted region has all other special edges on one side. Then we have m recursive steps with at most n, $n-2$, …, $n-2m+2$ points. The most amount of time we need to spend in each step is if the special edge is a local maximum. In this situation we then need to check whether the number of colors to the left and to the right is the same and possibly rename the colors on one side. This then takes at most $n-2i$ time in the ith step. As a result, the total time to construct a color assignment as seen in the proof for Lemma 7.9 is $O(nm)$.

7.3.3 The Matching Lower Bound

Similarly to the previous section we can combine the constructions for the logarithmic and linear lower bounds. The resulting input point set and input edge set will then need exactly as many colors as shown by the upper bound in Lemma 7.9. See Figure 7.6 for an example construction for $k = 1$ and $\ell = 2$ as well as $k = 2$ and $\ell = 1$.

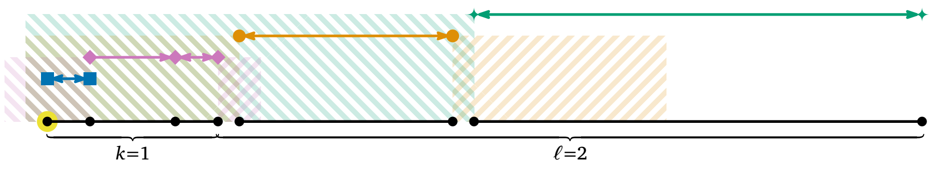
Lemma 7.10. *For all $k, \ell \in \mathbb{N}$ there exists an input edge set $E \subseteq E_P$ with k local maxima and ℓ small gaps for an input point set $P \subseteq \mathbb{R}$ with $n = 2(k+\ell+1)$ points such that every valid color assignment needs $1 + \lfloor \log(k+1) \rfloor + \ell$ colors.*

Proof. If $\ell = 0$ we refer to Lemma 7.4 and if $k = 0$ we refer to Lemma 7.8 to see that the claim already holds.

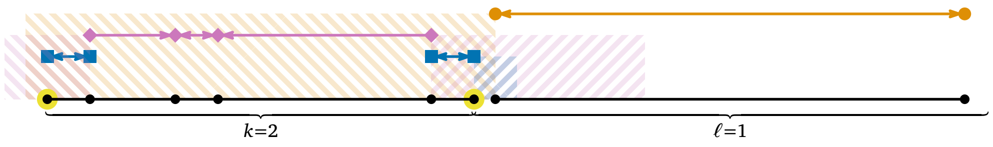
If $\ell, k > 0$ we first look at the proof of Lemma 7.4 and take the point set constructed there for $n = 2k+2$ with k local maxima and call it P. We know then that we already need $1 + \lfloor \log(k+1) \rfloor$ colors for any valid color assignment that results in $E = E_P$. The next step is to extend P and E with 2ℓ points and ℓ small gaps such that every added gap adds the necessity for an additional color. Here we take the idea from the proof of Lemma 7.8.

What we do is add a small gap to the right side of P such that its left restricted region spans P completely. Assume that the rightmost point of P is at position p. Since the smallest possible distance from p to its next point to the left is 1, we can just add a new point at position $p + 0.5$ to P. This way we can ensure that P continues to be an input

↪



(a) With $k = 1$ and $\ell = 2$ we need $1 + \lfloor \log(k + 1)\rfloor = 2$ colors for the left part and then additional $\ell = 2$ colors for the right part.



(b) With $k = 2$ and $\ell = 1$ we still need $1 + \lfloor \log(k + 1)\rfloor = 2$ colors for the left part and $\ell = 1$ additional color for the right part.

Figure 7.6: Two different lower bound constructions with different values for k and ℓ and one possible valid color assignment for each. In both figures we can see the marked points as those overlapped by the maximum number of restricted regions with different colors. For (a) it is three regions showing that four colors are needed, while for (b) it is two regions showing that three colors are needed.

point set. The next point is then added at position $2p + 1.5$ which means that for all points to the left of it, an equidistant point to the left would have negative coordinates. Since we know that no points with negative coordinates are in P, this ensures that P continues to be an input point set. We also add the edge $\{p + 0.5, 2p + 1.5\}$ to E. We can repeat this process until we have added 2ℓ points and with it ℓ small gaps. With the same arguments as in the proof of Lemma 7.8 we see that each small gap needs a new color.

As a result, P has $n = 2k + 2 + \ell$ points and E contains k local maxima and ℓ small gaps. The number of colors we need is $1 + \lfloor \log(k + 1)\rfloor + \ell$. □

7.4 Conclusion

In this chapter we have looked at the optimal number of colors for the 1D-CNNG and the 1D-CNNG-Gaps problem. We can combine the results from the different sections into the following all-encompassing

Theorem 7.1. *Let $P \subseteq \mathbb{R}$ be an input point set with n points and $E \subseteq E_P$ an input edge set with m special edges from which k are local maxima and ℓ are small gaps. Then $1 + \lfloor \log(k + 1)\rfloor + \ell \leq m + 1$ colors are always sufficient and sometimes necessary to* ↪

solve the 1D-CNNG-Gaps *problem for E. Relative to n we know that $\lfloor n/2 \rfloor$ colors are always sufficient and sometimes necessary.*

Looking at the 1D-CNNG *problem, it can be solved for P with at most $1 + \lfloor \log(k+1) \rfloor$ colors and some inputs need this many colors. Relative to n we know that $\lfloor \log n \rfloor$ colors are always sufficient and sometimes necessary.* □

We have also seen in Section 7.3.2 that we can find an actual BCA for any given input in $O(nm)$ time. However, the only guarantees for the number of colors in this BCA is, that it has at most $1 + \lfloor \log(k+1) \rfloor + \ell$ colors. We have no optimality or approximation guarantee.

Chapter 8

A Dynamic Program for a Fixed Color Number

So far we have shown how to solve the 1D-CNNG-Gaps problem in linear time for up to two colors in Chapter 6 and also how many colors are needed in the worst case in Chapter 7. In this chapter we describe an algorithm using a dynamic programming approach that solves the problem for an arbitrary large number of colors.

We can reuse many of the ideas that we developed in Chapter 6, especially the observations regarding BCAs in Section 6.3. We already know that for any color assignment with $\hat{c}$ colors there is also a BCA with the same number of colors, as shown in Lemma 6.5. Thus, we will limit the dynamic program to finding BCAs.

In this chapter we will first look at the size of the solution space in which we must search for a possible solution. We will see that this size is exponential in the number of special edges, meaning that a naive algorithm will need at least this amount of time to find a solution. We will then use some observations to design a dynamic program that is only exponential in the number of allowed colors. Looking at the results from the previous chapter we will see that this is still exponential in the number of special edges for arbitrary input. For inputs without gaps this will improve the running time to being exponential in the logarithm of the number of points.

8.1 A Quick Look at the Naive Approach

Before we dive into the dynamic program, we first want to think about the running time of a naive algorithm that enumerates all possible BCAs. Assume that we are given an input point set with n points and an input edge set that has m special edges out of which k are local maxima. Furthermore, assume that we are given $\hat{c}$ colors to use. To find a BCA with $\hat{c}$ colors there are basically two things to consider: The first is that for each local maximum we need to choose whether the left or the right endpoint gets two colors. These are 2^k possible combinations.

The second thing to consider is which color is assigned to which block. We have made similar observations at the beginning of Section 6.4 with the constraint that we had only two colors and thus by the change of color the actual color was forced. In our situation we have more colors and can thus choose which one to pick for each of the $m+1$ blocks. We can fix the first two blocks to two specific and distinct colors, and we know that we cannot repeat the second color in the third block. Thus, the third block can be given any of the remaining $\hat{c}-1$ colors. The same is true for every next block afterwards. This results in $(\hat{c}-1)^{m-1}$ different combinations of colors. In total there are then $2^k(\hat{c}-1)^{m-1}$ different BCAs with the first two blocks fixed to two specific colors.

If we were to implement this naive approach, one way could be to enumerate all possible BCAs. For each such BCA we would then need to compute its CNNM or check whether any of the conditions regarding the restricted regions are violated. Testing whether a specific BCA is valid can be done in time $O\left(m^2\right)$ by checking all pairs of special edges whether they and their colors make the BCA invalid. Then, the overall running time for a naive approach would be

$$O\left(m^2 2^k \hat{c}^{m-1}\right). \tag{Naive}$$

We know from Observation 6.3 that both k and m can be bounded by $\lfloor n/2 \rfloor - 1$ to express the above bound in terms of n, giving

$$O\left(n^2 2^{\frac{n}{2}} \hat{c}^{\frac{n}{2}-2}\right) = O\left(n^2 (2\hat{c})^{\frac{n}{2}-2}\right).$$

Using the upper bounds on the color numbers from Theorem 7.1 we can then also upper bound $\hat{c}$ and obtain

$$O\left(n^2 (2\log n)^{\frac{n}{2}-2}\right) \quad \text{and} \quad O\left(n^{\frac{n}{2}}\right)$$

as running times for the 1D-CNNG problem and the 1D-CNNG-Gaps problem, respectively.

8.1.1 Incremental construction

Instead of enumerating all BCAs, a different way would be to incrementally build BCAs from the left to the right. We could start by giving color 1 to the first block. If u_1 is a local maximum we would then have two possibilities to choose u_1's left or right endpoint as the one with two colors. We then assign color 2 to the second block. For the third block we then have exactly $\hat{c}-1$ different choices and also two choices for the endpoint of u_2 if it is a local maximum. If we branch our program off for every possible choice, we would create an exponential number of calculation branches, the same number as possible BCAs. We can improve this approach by throwing away an unfinished color assignment once we see that it is missing an edge or has duplicate or multiple edges. However, this would not improve the worst-case running time of the algorithm.

8.2 The Idea of and Prerequisites for the Dynamic Program

The main problem of the naive incremental approach in Section 8.1.1 is that we keep all the information about which blocks are colored in which color and which local maximum chose the left or the right endpoint as the one for two colors. However, as we will now show, we will only need to keep track of the closest special edge in each color. In the idea described in Section 8.1.1 we construct many BCAs in parallel, but they are all constructed from left to right. We can make this more formal with the following

▷ partial basic color assignment (partial BCA)
▷ partially valid
▷ extension
▷ completion

Definition 8.1. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$ be an input point set, $E \subseteq E_P$ an input edge set, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the set of special edges in E. Let now $u_i = \{p_a, p_{a+1}\} \in \mathscr{S}_E$ be a special edge. Then $P_i = \{p \in P \mid p \leq p_a\}$ is the input point set restricted to those points to the left of u_i including its left endpoint p_a. Furthermore, $E_i = E \cap E_{P_i} = \{\{p, q\} \in E \mid p \in P_i \wedge q \in P_i\}$ is the input edge set restricted to edges between the points in P_i.

Now, let $\sigma_i : P \to \mathbb{C}_{\hat{c}} \cup \{\varnothing\}$ be a function such that $\sigma_i(p) \neq \varnothing$ for all $p \in P_i$ and $\sigma_i(p) = \varnothing$ for all $p \notin P_i$. We call σ_i a *u_i-partial basic color assignment (partial BCA)* if it is a BCA for E_i, and we say that σ_i is *partially valid* for E (or just *partially valid* if E is clear from context) if σ_i is valid for E_i.

A u_j-partial BCA σ_j for some $j > i$ is called an *extension* of σ_i if $\sigma_i(p) = \sigma_j(p)$ for all $p < p_a$ and $\sigma_i(p_a) \subseteq \sigma_j(p_a)$. Similarly, we call a BCA σ a *completion* of σ_i if $\sigma_i(p) = \sigma(p)$ for all $p < p_a$ and $\sigma_i(p_a) \subseteq \sigma(p_a)$.

Thus, a u_i-partial BCA is a function where all points to the left of special edge u_i, including its left endpoint, are assigned colors and all points to the right of u_i (including its right endpoint) have no colors assigned. In addition, if we discard all points that have no colors assigned, the partial BCA must be a BCA for the remaining edges of the remaining points.

If we take a BCA and successively remove the color given to the rightmost block from the points in the rightmost block we obtain a special sequence of partial BCAs as defined by

▷ partial sequence

Definition 8.2. Let $E \subseteq E_P$ be an input edge set, $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the set of special edges and $\sigma : P \to \mathbb{C}_{\hat{c}}$ a BCA for E. For every $u_i = \left\{p_{a_i}, p_{a_i+1}\right\} \in \mathscr{S}_E$ let c_i be the color assigned to the block to its left (the one that contains p_{a_i}). We then define a function $\sigma_i : P \to \mathbb{C}_{\hat{c}} \cup \{\varnothing\}$ for u_i as follows:

$$\sigma_i(p) = \begin{cases} \sigma(p) & \text{if } p < p_{a_i}, \\ \{c_i\} & \text{if } p = p_{a_i}, \text{ and} \\ \varnothing & \text{otherwise } \left(p \geq p_{a_i+1}\right). \end{cases}$$

We call the sequence $\sigma_1, \dots, \sigma_m$ the *partial sequence* for σ.

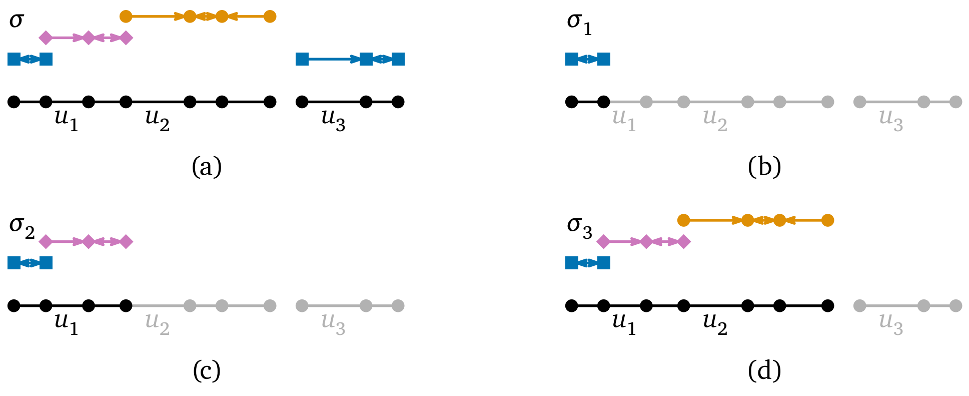


Figure 8.1: A valid BCA σ_i in (a) and its partial sequence σ_1, σ_2, σ_3 in (b–d). The uncolored points are grayed out to observe that the partial BCAs are partially valid.

See Figure 8.1 for an example BCA with its partial sequence. For every $u_i \in \mathscr{S}_E$ we can see that all colors assigned by σ_i are either exactly the colors assigned by σ or a subset of those colors. In addition, the rightmost point that has a color assigned by σ_i is the left endpoint of u_i and it has only one color assigned. Thus, the next observation follows immediately:

Observation 8.1. *Let $\sigma_1, \dots, \sigma_m$ be the partial sequence for a BCA σ with m special edges. Then σ_i is a u_i-partial BCA for all $1 \leq i \leq m$. Additionally, for two values $1 \leq i < j \leq m$ it holds that σ_j is an extension of σ_i. Furthermore, σ is a completion of every σ_i in the sequence.* □

What will be important for the dynamic program is the following: If there is a valid BCA for a given input edge set then all partial BCAs in the partial sequence are partially valid. See Figure 8.1 again for an example partial sequence where we can see that the partial BCAs are all partially valid.

Lemma 8.1. *Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the special edges. Let furthermore $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a valid BCA. Then all partial BCAs σ_i in the partial sequence for σ are partially valid.*

Proof. For every $u_i = \{p_a, p_{a+1}\} \in \mathscr{S}_E$ we show that σ_i is partially valid. Let $P_i = \{p \in P \mid p \leq p_a\}$ be the input point set restricted to points that have a color in σ_i and $E_i = E \cap E_{P_i}$ be the input edge set of the edges between the points in P_i. We can now easily see that σ_i is valid for E_i: For all special edges $u_j \in \mathscr{S}_{E_i}$ in E_i we know that no point with the wrong color is in any of the relevant restricted regions for u_j. Since the colors given to the endpoints of u_j are the same in σ_i and σ we need to check the same restricted regions for the same colors. The colors assigned by σ_i are subsets of those ↪

assigned by σ for all points in P_i and thus there can be no point with a wrong color in any of the relevant restricted regions for u_j regarding σ_i. Then σ_i is valid for E_i and thus σ_i is partially valid for E. □

As mentioned already before, the main problem of the naive approach of constructing all possible BCAs is that we keep track of every choice of color and endpoint with two colors so far. We will now see that we do not need all this information to extend a partially valid u_i-partial BCA to a partially valid u_{i+1}-partial BCA.

Lemma 8.2. *Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$, and $\mathscr{S}_E = \{u_1, \ldots, u_m\}$ be the special edges. For some $u_i \in \mathscr{S}_E$, let $\sigma_i : P \to \mathbb{C}_{\hat{c}}$ be a partially valid u_i-partial BCA. Furthermore, let c_l be the color shared by the rightmost block that has colors assigned by σ_i which is block B_{i-1}, and let $c_r \in C_{\hat{c}} \setminus \{c_l\}$ be some other color.*

If $i < m$, let $\sigma_{i+1} : P \to \mathbb{C}_{\hat{c}}$ be an extension of σ_i such that c_r is the color shared by the rightmost block that has colors assigned by σ_{i+1} which is B_i. We can then determine whether σ_{i+1} is partially valid by knowing two things from σ_i:

1. *The rightmost special edge u_j which has c_r as the color for its left block B_{j-1}, and*
2. *if u_j is a local maximum, which of the endpoints of u_j has two colors.*

If $i = m$, let $\sigma : P \to \mathbb{C}_{\hat{c}}$ be a completion of σ_i such that c_r is the color shared by the rightmost block for E which is B_m. We can then determine whether σ is valid with the same information from σ_i as above.

Proof. We start by looking at the case $i < m$, and we first note that if u_i is a small gap then there is only one possible extension σ_{i+1} while there are two possible extensions if u_i is a local maximum. In order to check whether σ_{i+1} is partially valid we need to check whether it is valid for $E_{i+1} = \{\{p, q\} \in E \mid p \leq p_a \wedge q \leq p_a\}$ with $u_{i+1} = \{p_a, p_{a+1}\}$. Since we know that σ_i is partially valid, we only need to look at the difference between σ_i and σ_{i+1}, namely B_i and the endpoints of u_i. From Lemma 6.7 we know that we only need to check whether any restricted regions contain a point with the wrong color.

In the statement of the lemma we say that u_j is the rightmost special edge that has color c_r in the block to its left in σ_i. It is, of course, not always the case that this special edge actually exists. If no point is assigned c_r by σ_i then u_j does not exist. For now, however, we assume that u_j exists and talk later about the case in which it does not.

If we look at u_i we see that its right restricted region can only contain points that have color c_r and thus no point with color c_l. However, its left restricted region may contain a point with color c_r which would make σ_i not valid if u_i is a small gap or its left endpoint has two colors. To check this, it is sufficient to know the rightmost point p with color c_r. This can be obtained from the two pieces of information that we claimed we needed. Then p is either u_j's left endpoint if u_j is a small gap or the endpoint with

↪

two colors if u_j is a local maximum. We can then conclude that σ_{i+1} is not valid if $p \in \overleftarrow{R}_E(u_i)$ unless u_i is a local maximum with two colors for its right endpoint.

Finally, we also need to check whether the new points with color c_r are inside any of the right restricted regions of special edges to the left of u_i. However, we only need to look at those special edges whose left color is c_r and which do not have its left endpoint with two colors. We know that u_j has c_r to its left and if it does not have its left endpoint with two colors we check whether the leftmost point of u_i with color c_r is inside $\overrightarrow{R}_E(u_j)$. If this is the case then σ_{i+1} is not valid.

We do not need to check any other special edges that are to the left of u_j since we know that σ_i is partially valid. Assume that there is a special edge to the left of u_j for which we need to check that no point in its right restricted region contains color c_r. If this right restricted region contained an endpoint of u_i it would also contain both endpoints of u_j and at least one of them has color c_r. Thus, already σ_i would not be valid.

So far we have assumed that a special edge u_j exists. If u_j does not exist, we know that no point is given color c_r by σ_i. This means that we do not need to perform any checks and we always know that σ_{i+1} is valid for E_{i+1} and thus partially valid for E.

We can see that we have not used the fact that σ_{i+1} is a partial BCA or that there exists a special edge u_{i+1} anywhere except when we defined E_{i+1}. If we replace E_{i+1} with E we can use all the same arguments and conclude that for the case in which $i = m$ we can check whether σ is valid for E with the information provided. □

We can now see that it is sufficient to know about the rightmost special edge with color c to its left for all colors $c \in C_{\hat{c}}$ to obtain a partially valid extension or a valid completion of a given partial BCA. That is, we only need one piece of information instead of arbitrarily many for each color. With this knowledge we can finally start to define our dynamic program. The idea is now that with the limited information passed to our decision-making, this information will be repeated in other parts of the calculation for different partially valid partial BCAs. Thus, by computing the information once and reusing it afterwards the running time of the upcoming dynamic program should be lower than the running time we saw for the naive approach.

8.3 Defining the Dynamic Program

For the remainder of this chapter assume now that we are given an input edge set $E \subseteq E_P$ for an input point set $P \subseteq \mathbb{R}$ with n points, and a number of colors $\hat{c}$. Let $\mathscr{M}_E$, $\mathscr{G}_E^-$, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the local maxima, small gaps, and special edges, respectively.

Overall, the principle underlying the dynamic program will be as follows: We start with one or two u_2-partial BCAs σ_2 (depending on whether u_1 is a local maximum or small gap) with the colors of the first two blocks already fixed to the first two colors. For each σ_2 we then try all possible partially valid u_3-partial BCAs σ_3 that are extensions of the σ_2. For each of those we try all possible partially valid u_4-partial BCAs σ_4 that are extensions of the respective σ_3, and so on. In the end we will find all possible valid BCAs that are completions of the partially valid u_m-partial BCAs.

In each step of the process we need to be given the special edge u_i which should be colored, i.e., the special edge for which we already have the u_i-partial BCA σ_i. We must also know which color we should use for the extension. And finally, as observed before for each color we need to be given the rightmost endpoint to the left of u_i that has this color assigned to at least one of its endpoints in σ_i. We also need to know on which side u_i has two colors if it is a local maximum.

8.3.1 Specification

To summarize this more clearly, the following information will be passed to each call of the dynamic program:

1. The special edge $u_i \in \mathscr{S}_E$ which should be colored now together with the block B_i to its right. In other words, u_i is the special edge for which we already have a u_i-partial BCA.
2. A color $\vec{c} \in C_{\hat{c}}$ telling us with which color the points to the right of u_i, i.e., those in B_i, should be colored.
3. For every color $c \in C_{\hat{c}}$ we pass two additional values:
   a) The rightmost special edge $s_c = u_j \in \mathscr{S}_E$ to the left of (and including) u_i such that the shared color in B_{j-1} is c. This will be u_i for the color that is directly to the left of u_i. If no block so far has been given color c, we provide the placeholder $\perp$. For convenience, we define $\mathscr{S}_{E,\perp} = \mathscr{S}_E \cup \{\perp\}$ as the set of special edges including $\perp$.
   b) A binary value $b_c \in \{L, R\}$ telling us which endpoint of s_c is the rightmost one to have color c. Thus, it is L if s_c is a small gap or if s_c has two colors at its left endpoint and R if s_c has two colors at its right endpoint. In the case that $s_c = \perp$ we set $b_c = L$.

As a result, the signature of the dynamic program is defined as

$$S : \mathscr{S}_E \times C \times \mathscr{S}_{E,\perp}^{\hat{c}} \times \{L, R\}^{\hat{c}} \to \{0, 1\},$$

and a call to

$$S(u_i, \vec{c}, \mathbf{s} = (s_1, \dots, s_{\hat{c}}), \mathbf{b} = (b_1, \dots, b_{\hat{c}}))$$

should then tell us whether there is a valid a BCA that is a completion for the u_i-partial BCA for which we only now the information passed as $\mathbf{s}$ and $\mathbf{b}$ with the following constraints:

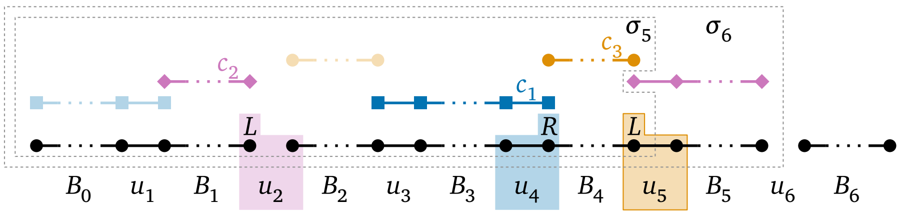


Figure 8.2: An abstract representation of a u_5-partial BCA σ_5 (the innermost dashed part) that should be extended to a u_6-partial BCA σ_6 (the outer dashed part). Here we only represent the special edges and the blocks without the points in the blocks. For σ_5 we can see that B_0 and B_3 are both given c_1, B_2 and B_4 are given c_3, and B_1 is given c_2. We marked u_2, u_4, and u_5 because those are the rightmost special edges with their respective color to the left. We also specified whether its left or right endpoint is the rightmost point with the color to its left. Assuming we have four colors in total, this situation is equivalent to calling the dynamic program as $S(u_5, c_2, (u_4, u_2, u_5, \bot), (R, L, L, L))$.

1. The shared color in block B_i to the right of u_i is $\vec{c}$.

2. We can extract the color $\overleftarrow{c}$ to the left of u_i from the fact that for exactly one color $\overleftarrow{c}$ we will have that $s_{\overleftarrow{c}} = u_i$. If u_i is a local maximum we then give two colors to its left or right endpoint depending on the value in $b_{\overleftarrow{c}}$.

3. For all colors $c \in C_{\hat{c}}$ such that s_c is a small gap or $b_c = R$ we ensure that no point inside $\overrightarrow{R}_E(s_c)$ is given color c.

See Figure 8.2 for an example u_5-partial BCA which should be extended to a u_6-partial BCA using color c_2. The figure represents a situation in which the dynamic program is called as $S(u_5, c_2, (u_4, u_2, u_5, \bot), (R, L, L, L))$. This implies that we allow the dynamic program to use up to four colors. We can see that the information that colors c_1 and c_3 are used for blocks B_0 and B_2, respectively, is not passed to the dynamic program. For all colors that are used (all but c_4) we have the rightmost special edge with an endpoint in this color and whether the right endpoint has this color or not.

We now of course want to know how the dynamic program works exactly, i.e., what actions are performed when $S(u_i, \vec{c}, \mathbf{s}, \mathbf{b})$ is called. However, before doing so, we first look at some useful functions that will help us describe it more succinctly.

8.3.2 Useful Helper Functions

We have specified that for every color $c \in C_{\hat{c}}$ we are given the rightmost special edge s_c such that c is given to its left block. This means that at least one of its endpoints has color c, but it may be both. For this we are given b_c which tells us whether it is only the left endpoint $b_c = L$ or both $b_c = R$. As observed in Lemma 8.2, in order to check whether a partial BCA is partially valid, in some cases we need to check whether the rightmost point with color $\vec{c}$ is inside $\overleftarrow{R}_E(u_i)$. In order to easily access this point we change L and R from being simple placeholders for the left and right endpoint into functions that, when applied to a special edge, return this endpoint. If applied to a $\perp$ value, they should just return $\perp$ again. That is, we define $L, R : \mathcal{S}_{E,\perp} \to P \cup \{\perp\}$ as

$$L(u_j) = \begin{cases} \perp & \text{if } u_j = \perp, \\ p_b & \text{if } u_j = \{p_b, p_{b+1}\}, \end{cases}$$
$$R(u_j) = \begin{cases} \perp & \text{if } u_j = \perp, \\ p_{b+1} & \text{if } u_j = \{p_b, p_{b+1}\}. \end{cases}$$

Now, whenever we want to check whether the rightmost point with a specific color $c \in C_{\hat{c}}$ is in the left restricted region of u_i we can just check $b_c(s_c) \in \overleftarrow{R}_E(u_i)$. If s_c is a special edge, b_c will return the correct endpoint. Very conveniently, if $s_c = \perp$ then b_c returns $\perp$ and the check will always return false, which is exactly what we want.

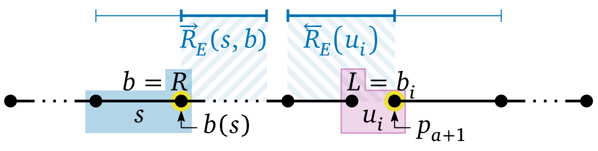
Another check that needs to be made is whether the leftmost endpoint of u_i that is given color $\vec{c}$ is inside the right restricted region of $s_{\vec{c}}$. However, we only need to make this check if $s_{\vec{c}}$ is a small gap or a local maximum with two colors at its right endpoint. To simplify the checks we overload the function $\overrightarrow{R}_E$ as $\overrightarrow{R}_E : \mathcal{S}_{E,\perp} \times \{L, R\} \to \mathbb{I} \cup \varnothing$. This means that we pass both a special edge or $\perp$ and the binary indicator for the left or right endpoint. The result can either be a restricted region or the empty set which will be returned whenever we do not need to check the special edges right restricted region. We define our overloaded version as

$$\overrightarrow{R}_E(s, b) = \begin{cases} \overrightarrow{R}_E(s) & \text{if } s \in \mathcal{G}_E^- \text{ or } s \in \mathcal{M}_E \wedge b = R, \\ \varnothing & \text{if } s = \perp \text{ or } s \in \mathcal{M}_E \wedge b = L. \end{cases}$$

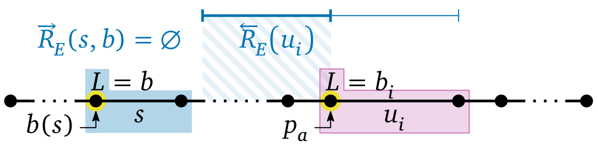
We can now just check $p \in \overrightarrow{R}_E(s_c, b_c)$ for some point $p \in P$ and some color $c \in C_{\hat{c}}$ and due to $\overrightarrow{R}_E$ returning the empty set whenever no right restricted region needs to be checked, we have exactly the desired behavior.

In every step of the dynamic program we will need to check both the potential right restricted region of a previous special edge and the potential left restricted region of the current special edge. As mentioned before, we want to be able to describe the dynamic program as succinctly as possible. We therefore move this check into its own verification function $V : \mathcal{S}_E \times \{L, R\} \times \mathcal{S}_{E,\perp} \times \{L, R\} \to \{0, 1\}$ which is defined as

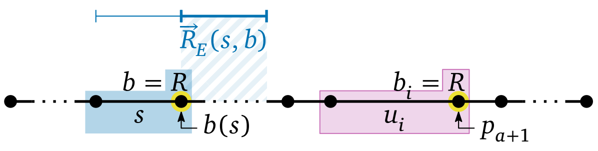
$$\text{let } u_i = \{p_a, p_{a+1}\} \text{ in}$$
$$V(u_i, b_i, s, b) = \begin{cases} p_{a+1} \notin \overrightarrow{R}_E(s, b) \wedge b(s) \notin \overleftarrow{R}_E(u_i) & \text{if } u_i \in \mathcal{G}_E^-, \\ p_a \notin \overrightarrow{R}_E(s, b) \wedge b(s) \notin \overleftarrow{R}_E(u_i) & \text{if } u_i \in \mathcal{M}_E \wedge b_i = L, \text{ and} \\ p_{a+1} \notin \overrightarrow{R}_E(s, b) & \text{if } u_i \in \mathcal{M}_E \wedge b_i = R. \end{cases}$$



(a) Since u_i is a small gap and $b = R$ we need to check whether u_i's right endpoint is inside the right restricted region of s. We also need to check whether u_i's left restricted region contains the right endpoint of s (because $b = R$).



(b) Since $b = L$ and s is a local maximum we do not need to check whether the right restricted region of s contains any point. This is made easy by $\vec{R}_E(s,b)$ evaluating to $\varnothing$. However, since $b_i = L$ we need to check whether u_i's left restricted region contains the left endpoint of s (because $b = L$).



(c) Since $b_i = R$, we do not need to check u_i's left restricted region. However, since $b = R$ and s is a local maximum we need to check whether the right restricted region of s contains the right endpoint p_{a+1} of u_i (since $b = R$).

Figure 8.3: The three different situations in the verification function: (a) when u_i is a small gap, (b) when it is a local maximum and its left endpoint should have two colors, and (c) when it is a local maximum and its right endpoint should have two colors. We have highlighted the endpoints for which we may need to check whether they are contained in the restricted regions of the other special edge.

The function is passed the special edge u_i for which we want to check whether we can color the points to its right in a specific color $\vec{c}$. We are also passed b_i which, in case u_i is a local maximum, tells us which of u_i's endpoints is chosen to have two colors. Additionally, we are given a rightmost special edge s and its binary indicator b. For the latter two values we will pass $s_{\vec{c}}$ and $b_{\vec{c}}$ in the dynamic program. We now break down what the function does. For each of the three parts of the function we refer to Figure 8.3 for example situations.

If u_i is a small gap (first line), we know that its right endpoint is the only one with the new color $\vec{c}$. Thus, we check whether this endpoint is inside the right restricted region of the passed special edge s. Additionally, we need to check whether u_i's left restricted region contains the rightmost point of s with color $\vec{c}$. This point is obtained by using b as a function on s as described before. Note that, due to the definition of the helper functions, whenever $s = \perp$ both statements are trivially true. See Figure 8.3a for an example with $b = R$, i.e., where we need to check whether the right restricted region of s contains the right endpoint of the small gap and whether the left restricted region of u_i contains $b(s)$.

If u_i is a local maximum and its left endpoint should have two colors we know that this left endpoint will have color $\vec{c}$ and thus check whether it is insides the right restricted region of s. Additionally, we also need to check whether the rightmost point of s with color $\vec{c}$ is inside u_i's left restricted region. See Figure 8.3b for an example. Here we have depicted a situation where $b = L$ and thus $\overrightarrow{R}_E(s, b) = \varnothing$.

Finally, if u_i is a local maximum and its right endpoint should have two colors we know that for u_i we only need to check its right restricted region. Thus, the only check we perform here is to see whether u_i's right endpoint is inside the right restricted region of s. In Figure 8.3c we see an example situation.

Note that in the previous description we have used $\vec{c}$ in the description, even though it is not passed to V. However, as described before, V will only be called with specific parameters for which we can make those assumptions. The color is also only used to help our understanding of V's functionality, it is unimportant for its definition.

8.3.3 Definition

We are now finally ready to define S, the actual function for the dynamic program. As S will be recursively defined, we start with the base case, which is the situation when $u_i = u_m$, i.e., we are passed the rightmost special edge. In this case we know that we have already found a partially valid u_m-partial BCA σ_m. The only question is then whether giving the passed color $\vec{c}$ to the rightmost block will lead to a valid completion σ of σ_m or not. Thus, no further recursive calls are needed and the only task is to check whether the new points with color $\vec{c}$ introduce any problems. However, we have already defined a helper function V to perform this specific check. We can thus define the base case as

$$S(u_m, \vec{c}, \mathbf{s}, \mathbf{b}) = V(u_m, b_{\overleftarrow{c}}, s_{\vec{c}}, b_{\vec{c}}). \tag{DP.B}$$

We remember that we can extract $\overleftarrow{c}$, the color shared in block B_{m-1}, by finding the special edge in $\mathbf{s}$ for which $s_{\overleftarrow{c}} = u_m$.

In the recursive case we will (again) first need to check whether assigning $\vec{c}$ to the points to the right of u_i is actually even possible; we can just reuse V to do it. Then, we try out all possible ways to color the points to the right of u_{i+1}. Here we come across a slight inconvenience: Depending on whether u_{i+1} is a small gap or a local maximum we need to try out one possibility per color or two (one for each endpoint). We thus define the recursive case in two steps, first for the case that u_{i+1} is a small gap:

$$S(u_i, \vec{c}, \mathbf{s}, \mathbf{b}) = V(u_i, b_{\overleftarrow{c}}, s_{\vec{c}}, b_{\vec{c}}) \wedge \bigvee_{c \in C_{\hat{c}} \setminus \{\vec{c}\}} S(u_{i+1}, c, \mathbf{s}[\vec{c} \leftarrow u_{i+1}], \mathbf{b}[\vec{c} \leftarrow L]). \tag{DP.G}$$

Here $\mathbf{s}[\vec{c} \leftarrow u_{i+1}]$ is the same tuple as $\mathbf{s}$ with the only difference at index $\vec{c}$ where we change the value to u_{i+1}. The same notation for $\mathbf{b}$ naturally has the same effect. Thus, in the recursive call it will be that $s_{\vec{c}} = u_{i+1}$ and $b_{\vec{c}} = L$.

According to the definition, S first uses V to verify whether we can actually give color $\vec{c}$ to block B_i. Then, for all colors c that are not $\vec{c}$ we check recursively whether we can give color c to block B_{i+1}. We need to pass the information that B_i is now given color $\vec{c}$ and that the rightmost point with this color is the left endpoint of u_{i+1} because it is a small gap.

For the second case in which u_{i+1} is a local maximum we will have basically the same definition with the small difference that we make two recursive calls per color:

$$S(u_i, \vec{c}, \mathbf{s}, \mathbf{b}) = V(u_i, b_{\overleftarrow{c}}, s_{\vec{c}}, b_{\vec{c}}) \wedge \bigvee_{c \in C_{\hat{c}} \setminus \{\vec{c}\}} \Big[S(u_{i+1}, c, \mathbf{s}[\vec{c} \leftarrow u_{i+1}], \mathbf{b}[\vec{c} \leftarrow L]) \vee S(u_{i+1}, c, \mathbf{s}[\vec{c} \leftarrow u_{i+1}], \mathbf{b}[\vec{c} \leftarrow R]) \Big] \quad \text{(DP.M)}$$

By passing once $b_{\vec{c}} = L$ and once $b_{\vec{c}} = R$ to the recursive calls we independently check whether it is possible to have two colors at the left or at the right endpoint of u_{i+1}.

We have now given a formal definition of the dynamic program via the base case and two disjoint recursive cases. It remains to see how the dynamic program will be called from the outside to obtain the overall solution.

8.3.4 Calling the Dynamic Program

Now that we know how the dynamic program is defined we need to see what the initial values are with which it is called. In the very beginning of Section 8.3 we mentioned that we can fix the two colors for the first two blocks. This is true because if there is a valid BCA then the two colors it assigns to the first two blocks must be different. Renaming those two colors in the BCA we would then obtain a BCA that had the exact same two colors for the first block that we fix.

When fixing the first two colors, there is the question about what to do with the first special edge u_1 between the first two blocks. If u_1 is a small gap there is only one possibility to color it, but if u_1 is a local maximum we again have two choices. We can thus define a binary value

$$S^* = \begin{cases} S\Big(u_1, 2, \bot^{\hat{c}}[1 \leftarrow u_1], L^{\hat{c}}\Big) & \text{if } u_1 \in \mathcal{G}_E^- \\ S\Big(u_1, 2, \bot^{\hat{c}}[1 \leftarrow u_1], L^{\hat{c}}\Big) \vee & \\ S\Big(u_1, 2, \bot^{\hat{c}}[1 \leftarrow u_1], L^{\hat{c}}[1 \leftarrow R]\Big) & \text{if } u_1 \in \mathcal{M}_E \end{cases} \quad \text{(DP.S)}$$

which is supposed to be 1 if and only if there exists a valid BCA for E. Here $\bot^{\hat{c}}$ is a tuple with $\hat{c}$ entries of the same value $\bot$; the same is true for $L^{\hat{c}}$. The definition then asks whether there is a valid BCA when giving color 2 to the second block while the only information we have is that the points to the left of u_1 are given color 1. If u_1 is a small gap, we know that it is u_1's left endpoint that is the rightmost point with color 1. If, however, u_1 is a local maximum we have the situation that we give two colors to its left endpoint in which case, same as for small gaps, its left endpoint is the rightmost with color 1. We also have to check the other opportunity that we give two colors to its right endpoint in which case we need to pass the information that its right endpoint is the rightmost one with color 1.

8.3.5 Extracting a Color Assignment

If $S^* = 1$ it means that there is a computation path starting at Eq. (DP.S) with a call to $S(u_1, \dots)$ and ending in Eq. (DP.B) with a call to $S(u_m, \dots)$. This computation path then has exactly one call to $S(u_i, \dots)$ for each special edge u_i. Every such call tells us at least which color is given to the points to the left of u_i (extracted from $\mathbf{s}$) and which color is given to the points on the right of u_i (parameter $\vec{c}$). From $\mathbf{b}$ we can also extract which endpoint of u_i has two colors if it is a local maximum. This information is enough to extract the BCA for one computation path.

How this computation path is obtained depends on the evaluation strategy. If we are only interested to find one BCA it would be sensible to compute the solution by evaluating S^* in a top-down fashion, employing memoization to prevent duplicate calculations. This way we can stop once we reach the first call to $S(u_m, \dots)$ that gives us a positive result and directly calculate the BCA while going back through the recursive calls. As a result, extracting the BCA would not take any additional time.

If we are interested in obtaining all possible BCAs, we could employ the same strategy with the difference that in each recursion step we maintain the corresponding partial BCA. Then, whenever we obtain a positive result from a call to $S(u_m, \dots)$ we can already report the associated BCA. A (partial) BCA can be seen as a list containing the color of a block alternating with which endpoint of a local maximum has two colors. Thus, maintaining the corresponding BCA only takes a constant amount of time since we only need to append two values to the list in each recursion step and remove them afterwards. Then, the only additional time we need when obtaining all BCAs is the time to report them which is $O(mN)$ in total where N is the number of BCAs.

8.4 Correctness and Running Time

We now only have to show that the program actually returns the correct result and see what its running time is.

8.4.1 Correctness

To show correctness for the dynamic program we first show that when S^* is evaluated we find all partially valid partial BCAs:

Lemma 8.3. *Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$, and let $\hat{c}$ be a number of colors. Let $\mathscr{M}_E$, $\mathscr{G}_E^-$, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the local maxima, small gaps, and special edges, respectively. Evaluating S^* will then find all partially valid u_i-partial BCAs (with the colors for the first two blocks fixed) for all $u_i \in \mathscr{S}_E$ and try out all of their possible extensions (and completions for $u_i = u_m$).*

Proof. We show this by induction on the index i of the special edge u_i.

Induction base $i = 1$. By definition a u_1-partial BCA is a color assignment that assigns a color only to the first block. Since a single block does not contain any small gaps or local maxima it can be colored by one color and thus all u_1-partial BCA are partially valid. We, however, only try out one u_1-partial BCA, namely the one which assigns color 1 to the points in the first block. For this we then try all possible extensions: If u_1 is a small gap there is only one u_2-partial BCA extension where the color of the second block is fixed to 2 and if u_1 is a local maximum there are two such u_2-partial BCA extensions. Those extensions are exactly the ones we try out in Eq. (DP.S).

Induction step $i \to i + 1$. When S is called for u_i the first thing we do is call the verification function V which checks whether we can apply the given color to the block B_i. If the result is positive, we know, according to Lemma 8.2, that this represents a partially valid u_{i+1}-partial BCA if $i < m$ and a valid BCA if $i = m$.

If $i < m$ we then make recursive calls for u_{i+1} for all possible colors to give to B_{i+1}, see Eq. (DP.G). In addition, if u_{i+1} is a local maximum we try both ways to give two colors to an endpoint of u_{i+1}, see Eq. (DP.M). These are all possible u_{i+2}-partial BCA extensions of the partially valid u_{i+1}-partial BCA. In case $u_{i+1} = u_m$ it is not the u_{i+2}-partial BCA extensions that we try but BCA completions.

If $i = m$ we are in the base case Eq. (DP.B) of the dynamic program and are finished after calling V. □

We can now use the previous result and Lemma 8.1 to show that the dynamic program will actually return the correct result.

Lemma 8.4. *Let $E \subseteq E_P$ be an input edge set for an input point set $P \subseteq \mathbb{R}$, and let $\hat{c}$ be a number of colors. Then $S^* = 1$ if and only if there is a valid BCA for E.*

Proof. Let $\mathscr{M}_E$, $\mathscr{G}_E^-$, and $\mathscr{S}_E = \{u_1, \dots, u_m\}$ be the local maxima, small gaps, and special edges, respectively. We first show that if a valid BCA exists, that then $S^* = 1$. Let σ be a valid BCA such that, without loss of generality, the colors of the first and second block are 1 and 2. Let $\sigma_1, \dots, \sigma_m$ be the partial sequence for σ. From Lemma 8.1 we know that for all $u_i \in \mathscr{S}_E$ the u_i-partial BCA σ_i is partially valid. According to Lemma 8.3 the dynamic program tries out all possible extensions and completions. This means that there is a calculation path in the dynamic program such that starting

↪

at σ_1 we first find σ_2, then σ_3, and so on. For all those partial BCAs we determine that they are partially valid and thus continue the recursion. Finally, we have found the partially valid u_m-partial BCA σ_m and try out all possible completions upon which me encounter σ. This will make this base case of the recursion return 1 which then makes $S^* = 1$.

Assume now that no valid BCA exists for the input, i.e., every BCA σ for E is not valid. Let $\sigma_1, \dots, \sigma_m$ be the partial sequence for any such σ. If σ_j is not partially valid for some $1 \leq j \leq m$ it follows that all σ_k with $k > j$ are not partially valid, as well: σ_j is not partially valid because there is a point with wrong color in a restricted region of a special edge. Both the point and the special edge will have the same colors assigned in σ_k as in σ_j, and thus cannot be partially valid.

For the partial sequence there must be either a value $1 \leq j < m$ such that σ_j is partially valid and σ_{j+1} is not, or σ_m is partially valid. This follows from the fact that σ_1 is always (trivially) partially valid. Then in σ_{j+1} or σ either of the following happens: One of the newly colored endpoints is in a right restricted region from a special edge to the left or the left restricted region of u_j or u_m contains a point with the wrong color. In the dynamic program we would find the last partially valid partial BCA σ_j or σ_m and identify that it is partially valid. Then we would try out all extensions for σ_j or completions for σ_m. For this we test whether the newly colored points are in the wrong right restricted region or whether the left restricted region of u_j or u_m contains a wrong point. Thus, the verification function V returns 0, and we return 0 for this computation branch. As a result, we do not return 0 for any of the computation branches, which in turn results in $S^* = 0$. □

8.4.2 Running Time

For the running time we need to look at all the time needed by S and all the helper functions that are used. We can easily see that V, the only helper function called by S, performs at most two checks whether a point is inside a restricted region. Both the involved points and the involved restricted regions are calculated in constant time. We thus see, that the verification checks with V are done in $O(1)$ time.

Assume that the results of all calls in the form $S(u_{i+1}, \dots)$ are accessible in constant time when calculating the result of a call in the form $S(u_i, \dots)$. Then, the most amount of time is spent when we are in the recursive situation of Eq. (DP.M) where we need to logically combine $2(\hat{c} - 1) + 1 = 2\hat{c} - 1$ values. In addition, we also need to extract color $\overleftarrow{c}$ from the information in $\mathbf{s}$ which also takes at most $O(\hat{c})$ time. This means that one call to S takes $O(\hat{c})$ time.

Let us now look at the number of possible parameter combinations with which S may be called.

1. For the first parameter u_i we have m different possible values. We can look at it in more detail and see, for example, that for u_1 we only have up to two different

calls, bringing this value down to $m-1$. This will, however, not matter for the asymptotic analysis.

2. The second parameter $\vec{c}$ can take $\hat{c}$ different values.

3. For the third parameter $\mathbf{s}$ we first observe that one of the $\hat{c}$ colors must have u_i as its special edge. As a result, there are only $\hat{c}-1$ other entries in $\mathbf{s}$ and they are either an edge different from u_i (since no block can have more than one color) or $\perp$. Thus, the number of possible arguments for the third parameter is $\hat{c}m^{\hat{c}-1}$. Here $\hat{c}$ represents the choice of color with u_i as special edge and $m^{\hat{c}-1}$ is an upper bound for distributing the other edges across the remaining colors and $\perp$.

4. Finally, the fourth parameter has at most $2^{\hat{c}}$ possible values.

   We can see, however, that if there are fewer local maxima than colors, i.e., $k < \hat{c}$, we can have at most 2^k possible choices for the fourth parameter. Since the third parameter then already defines which color has a small gap and which a local maximum we do not need to take the $\binom{\hat{c}}{k}k!$ possible distributions of local maxima onto positions into account. Thus, we can say that we have $2^{\min(\hat{c},k)}$ possible choices for the fourth parameter.

This gives an overall upper bound on the number of different calls to S of

$$m \cdot \hat{c} \cdot \hat{c}m^{\hat{c}-1} \cdot 2^{\min(\hat{c},k)} \in O\Big(\hat{c}^2 2^{\min(\hat{c},k)} m^{\hat{c}}\Big).$$

Together with the running time of $O(\hat{c})$ per call we have an overall running time for the dynamic program of

$$O\Big(\hat{c}^3 2^{\min(\hat{c},k)} m^{\hat{c}}\Big).$$

We will now briefly compare this running time with the one of the naive approach in Eq. (Naive). For our convenience we restate the running time of the naive approach which was

$$O\Big(m^2 2^k \hat{c}^{m-1}\Big).$$

Our main observation from comparing the two running times is that the dynamic program was able to bring the dependence on the number of special edges from the exponent into the base. On the other hand, the number of colors has gone from the base into the exponent. In our more detailed comparison in Section 8.5 we will see that for some situations the naive approach will be better than our dynamic program.

If we want to see how the running time looks with respect to the number of points in the input point set we can apply two bounds: We use the upper bound of $m \leq \lfloor n/2 \rfloor - 1$ from Observation 6.3 and the fact that $k \leq m$ and thus obtain a running time of

$$O\left(\hat{c}^3 2^{\min(\hat{c},\lfloor n/2 \rfloor - 1)} \left(\frac{n}{2}\right)^{\hat{c}}\right).$$

8.4.3 Conclusion

Using the correctness shown in Lemma 8.4 and the running time from the previous section we can bring it all together into one

Theorem 8.1. *Let $P \subseteq \mathbb{R}$ be an input point set with n points and $E \subseteq E_P$ an input edge set with m special edges from which k are local maxima and ℓ are small gaps. Given a color number $\hat{c}$, we can find a valid BCA for E or return that none exists in time*

$$O\Big(\hat{c}^3 2^{\min(\hat{c},k)} m^{\hat{c}}\Big) \subseteq O\bigg(\hat{c}^3 2^{\min(\hat{c},\lfloor n/2 \rfloor)} \Big(\frac{n}{2}\Big)^{\hat{c}}\bigg).$$

Let c^* be the optimal color number for the given input. We can then find c^* and thus a valid BCA for E with c^* colors in the same time as just mentioned with $\hat{c} = c^*$.

Bounding the optimal color number c^* first in terms of the local maxima k and small gaps ℓ and then in terms of the number of points n we obtain the following results: For the 1D-CNNG-Gaps *we can find the optimal color number and a valid BCA in time*

$$O\Big((1 + \lfloor \log(k+1) \rfloor + \ell)^3 2^{\min(1+\lfloor \log(k+1) \rfloor + \ell, k)} (k+\ell)^{1+\lfloor \log(k+1) \rfloor + \ell}\Big) \subseteq O\Big(n^{\lfloor n/2 \rfloor + 3}\Big).$$

For the 1D-CNNG *problem the bounds improve to*

$$O\Big((1 + \lfloor \log(k+1) \rfloor)^3 (2k)^{1+\lfloor \log(k+1) \rfloor}\Big) \subseteq O\Big(\log^3 n \cdot n^{\log n}\Big) \subseteq O\Big(\log^3 n \cdot 2^{\log^2 n}\Big).$$

Proof. We refer to Section 8.4.1 for the correctness of the algorithm. There and more specifically in Lemma 8.4 we show that $S^* = 1$ if and only if a BCA for the input exists. For the running time of

$$O\Big(\hat{c}^3 2^{\min(\hat{c},k)} m^{\hat{c}}\Big) \subseteq O\bigg(\hat{c}^3 2^{\min(\hat{c},\lfloor n/2 \rfloor)} \Big(\frac{n}{2}\Big)^{\hat{c}}\bigg)$$

we already gave the arguments in Section 8.4.2.

To find c^*, the smallest $\hat{c}$ for which we can find a valid BCA, we just start with $\hat{c} = 1$ and increase it by 1 every time the dynamic program does not find a solution. We then have a running time of

$$\begin{aligned} O\Bigg(\sum_{\hat{c}=1}^{c^*} \hat{c}^3 2^{\min(\hat{c},k)} m^{\hat{c}}\Bigg) &= O\Big((c^*)^3 2^{\min(c^*,k)} m^{c^*} + (c^*-1)^3 2^{\min(c^*-1,k)} m^{c^*-1} + \cdots + m\Big) \\ &= O\Big((c^*)^3 2^{\min(c^*,k)} m^{c^*}\Big), \end{aligned}$$

because asymptotically the running time for $\hat{c} = c^*$ dominates the running time for all previous runs with $\hat{c} < c^*$.

Using Theorem 7.1 we remember that for the 1D-CNNG-Gaps problem the optimal color number c^* is upper bounded by $1 + \lfloor \log(k+1) \rfloor + \ell$, giving the first upper bound. Since we can further upper bound c^* by $\lfloor n/2 \rfloor$ and also k and $k + \ell$ by $\lfloor n/2 \rfloor - 1$ we obtain

$$\begin{aligned}
&\big(1+\lfloor\log(k+1)\rfloor+\ell\big)^3 2^{\min(1+\lfloor\log(k+1)\rfloor+\ell,k)}(k+\ell)^{1+\lfloor\log(k+1)\rfloor+\ell}\\
&\quad\le \left(\left\lfloor\frac{n}{2}\right\rfloor\right)^3 2^{\min(\lfloor n/2\rfloor,\lfloor n/2\rfloor-1)}\left(\left\lfloor\frac{n}{2}\right\rfloor-1\right)^{\lfloor n/2\rfloor}\\
&\quad\le n^3 2^{\lfloor n/2\rfloor}\left(\frac{n}{2}\right)^{\lfloor n/2\rfloor}\\
&\quad= n^{\lfloor n/2\rfloor+3}.
\end{aligned}$$

For the 1D-CNNG problem we know that $\ell = 0$, thus $m = k$ and the bound on c^* is improved to $1 + \lfloor\log(k + 1)\rfloor$. We thus have

$$\begin{aligned}
&\big(1+\lfloor\log(k+1)\rfloor\big)^3 2^{\min(1+\lfloor\log(k+1)\rfloor,k)} k^{1+\lfloor\log(k+1)\rfloor}\\
&\quad= \big(1+\lfloor\log(k+1)\rfloor\big)^3 2^{1+\lfloor\log(k+1)\rfloor} k^{1+\lfloor\log(k+1)\rfloor} &&\text{for all } k\ge 2\\
&\quad= \big(1+\lfloor\log(k+1)\rfloor\big)^3 (2k)^{1+\lfloor\log(k+1)\rfloor}
\end{aligned}$$

giving us the first bound for the 1D-CNNG problem. With $k \le \lfloor n/2\rfloor - 1$ we can then simplify to

$$\begin{aligned}
&\big(1+\lfloor\log(k+1)\rfloor\big)^3 (2k)^{1+\lfloor\log(k+1)\rfloor}\\
&\quad\le \lfloor\log n\rfloor^3\left(2\cdot\frac{n}{2}\right)^{\lfloor\log n\rfloor}\\
&\quad\le \log^3 n\cdot n^{\log n}\\
&\quad= \log^3 n\cdot 2^{\log^2 n}.
\end{aligned}$$

□

8.5 Comparing the Dynamic Program and the Naive Approach

We will now compare the obtained running times for the dynamic program with the times observed before for the naive approach in Section 8.1. To do this we give names to the running times, starting with

$$T_n(k,l) = (k+\ell)^2 2^k \big(1+\lfloor\log(k+1)\rfloor+\ell\big)^{k+\ell-1}$$

for the naive approach and

$$T_d(k,l) = \big(1+\lfloor\log(k+1)\rfloor+\ell\big)^3 2^{\min(1+\lfloor\log(k+1)\rfloor+\ell,k)}(k+\ell)^{1+\lfloor\log(k+1)\rfloor+\ell}$$

for the dynamic program. We ignore constant factors since we will only compare the asymptotic behavior of both functions.

8.5.1 Running Times Without Small Gaps

For the 1D-CNNG problem, or more generally the 1D-CNNG-Gaps problem without small gaps, we see that we were able to significantly improve the running time. If $\ell = 0$ and thus $m = k$ we can see that for the running time of the naive approach we have

$$\begin{aligned} T_n(k,0) &= k^2 2^k (1 + \lfloor \log(k+1) \rfloor)^{k-1} \\ &\geq 2^k k^2 \log^{k-1} k \end{aligned} \tag{8.1}$$

while for the dynamic program (for $k \geq 2$) the running time is

$$\begin{aligned} T_d(k,0) &= \left(1 + \lfloor \log(k+1) \rfloor\right)^3 2^{1+\lfloor \log(k+1) \rfloor} k^{1+\lfloor \log(k+1) \rfloor} \\ &\leq \left(1 + \log 2k\right)^3 2^{1+\log 2k} k^{1+\log 2k} \\ &= \left(2 + \log k\right)^3 2^{2+\log k} k^{2+\log k} \\ &= 2^2 \left(2 + \log k\right)^3 k^3 k^{\log k}. \end{aligned}$$

Assuming that $k \geq 4$ we know that $2 \leq \log k$ and we can simplify to

$$\begin{aligned} &\leq 2^2 \left(2 \log k\right)^3 k^3 k^{\log k} \\ &= 2^5 k^{\log k} k^3 \log^3 k. \end{aligned} \tag{8.2}$$

Comparing Eqs. (8.1) and (8.2) will show us now that $T_d(k,0) \ll T_n(k,0)$:

$$\begin{aligned} (8.2) &< (8.1) \\ \Leftrightarrow 2^5 k^{\log k} k^3 \log^3 k &< 2^k k^2 \log^{k-1} k \\ \Leftrightarrow 2^5 k^{\log k} k &< 2^k \log^{k-4} k \\ \Leftrightarrow 5 + \log^2 k + \log k &< k + (k-4) \log k \\ \Leftrightarrow 5 + \log^2 k + 5 \log k &< k + k \log k. \end{aligned}$$

The latter inequality is true for all $k \geq 7$ and the dominating term $\log^2 k$ on the left-hand side grows much slower than the dominating term $k \log k$ on the right-hand side. As a result, we can see that for inputs without small gaps the dynamic program will outperform the naive approach by far.

8.5.2 Running Times Without Local Maxima

If, on the other hand, $k = 0$ and thus $\ell = m$, the running time for the naive approach simplifies to $T_n(0,\ell) = \ell^2(1+\ell)^{\ell-1}$ which can be upper and lower bounded as

$$\ell^{\ell+1} < T_n(0,\ell) = \ell^2(1+\ell)^{\ell-1} < (1+\ell)^{\ell+1}$$

and the running time for the dynamic program is simply $T_d(0,\ell) = (1+\ell)^3 \ell^{1+\ell}$ which can also be bounded on both sides as

$$\ell^{\ell+4} < T_d(0,\ell) = (1+\ell)^3 \ell^{1+\ell} < (1+\ell)^{\ell+4}.$$

It is thus easy to see that the naive approach is asymptotically faster by a factor of ℓ^3.

8.5.3 Running Times in Between

If one algorithm is faster for $k = 0$ and the other is faster for $\ell = 0$, then for all $m \in \mathbb{N}$ there must be values for k and ℓ such that both algorithms have (roughly) the same running time. However, due to the complexity of the running times we are unable to set $T_n(k,l) = T_d(k,l)$ and solve for k or l.

Slightly fewer small gaps than local maxima. One part that complicates the running time of the dynamic program is $\min\bigl(1 + \lfloor\log(k+1)\rfloor + \ell, k\bigr)$. Our approach is thus to look at the situation in which both parts in the minimum are equal. We thus set $1 + \lfloor\log(k+1)\rfloor + \ell = k$ and obtain

$$T_n(k,l) = (k+\ell)^2 2^k k^{k+\ell-1} = \bigl(2k - \lfloor\log(k+1)\rfloor - 1\bigr)^2 2^k k^{2k-\lfloor\log(k+1)\rfloor-2}$$

and

$$T_d(k,l) = k^3 2^k (k+\ell)^k = k^3 2^k \bigl(2k - \lfloor\log(k+1)\rfloor - 1\bigr)^k.$$

We claim that then $T_d(k,l) < T_n(k,l)$ for sufficiently large k:

$$\begin{aligned} & T_d(k,l) < T_n(k,l) \\ \Leftrightarrow\ & k^3 2^k \bigl(2k - \lfloor\log(k+1)\rfloor - 1\bigr)^k < \bigl(2k - \lfloor\log(k+1)\rfloor - 1\bigr)^2 2^k k^{2k-\lfloor\log(k+1)\rfloor-2} \\ \Leftrightarrow\ & \bigl(2k - \lfloor\log(k+1)\rfloor - 1\bigr)^{k-2} < k^{2k-\lfloor\log(k+1)\rfloor-5}. \end{aligned} \tag{8.3}$$

We can use $2k - \lfloor\log(k+1)\rfloor - 1 < 2k$ as an upper bound for the base on the left side while for the right side we see that $\lfloor\log(k+1)\rfloor \le 1 + \log k$. The latter thus gives a new lower bound for the right side of the inequality. As a result, if the following inequality holds, then Eq. (8.3) holds as well.

$$\begin{aligned} \Leftarrow\ & (2k)^{k-2} < k^{2k-\log k-6} \\ \Leftrightarrow\ & (k-2)\log(2k) < \bigl(2k - \log k - 6\bigr)\log k \\ \Leftrightarrow\ & k\log k + k - 2\log k - 2 < 2k\log k - \log^2 k - 6\log k \\ \Leftrightarrow\ & k + 4\log k + \log^2 k < k\log k + 2 \end{aligned}$$

which we can see is a true statement for large enough k, more precisely for $k \ge 10$. In Figure 8.4 we have also plotted the logarithms of the two sides from Eq. (8.3) before applying the bounds. There we can see that the inequality already holds for $k \ge 6$.

Same number of small gaps and local maxima. We now see what happens with the running times if $k = \ell$, that is, we have running times of

$$T_n(k,l) = (2k)^2 2^k \bigl(1 + \lfloor\log(k+1)\rfloor + k\bigr)^{2k-1}$$

for the naive approach and

$$\begin{aligned} T_d(k,l) &= \bigl(1 + \lfloor\log(k+1)\rfloor + k\bigr)^3 2^{\min(1+\lfloor\log(k+1)\rfloor+k,k)} (2k)^{1+\lfloor\log(k+1)\rfloor+k} \\ &= \bigl(1 + \lfloor\log(k+1)\rfloor + k\bigr)^3 2^k (2k)^{1+\lfloor\log(k+1)\rfloor+k} \end{aligned}$$

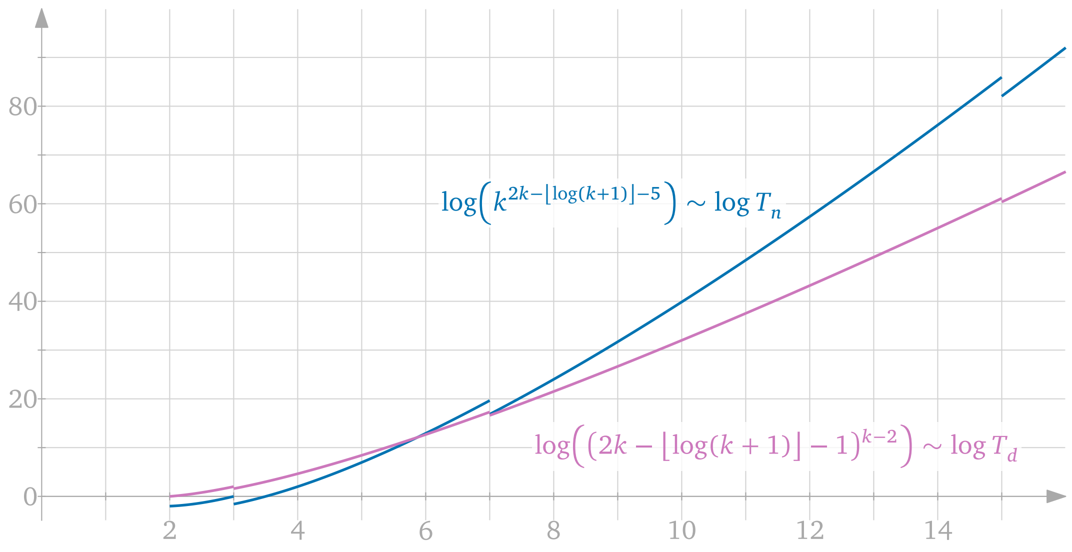


Figure 8.4: A plot of the logarithm of $(2k - \lfloor\log(k+1)\rfloor - 1)^{k-2}$ which is proportional to T_d and of $k^{2k-\lfloor\log(k+1)\rfloor-5}$ which is proportional to T_n.

for our dynamic program. We now show that for this situation the dynamic program is still better:

$$\begin{aligned}(1+\lfloor\log(k+1)\rfloor+k)^3 2^k (2k)^{1+\lfloor\log(k+1)\rfloor+k} &< (2k)^2 2^k (1+\lfloor\log(k+1)\rfloor+k)^{2k-1}\\ \Leftrightarrow (2k)^{\lfloor\log(k+1)\rfloor-1+k} &< (1+\lfloor\log(k+1)\rfloor+k)^{2k-4}.\end{aligned}$$

Here we can use the upper bound $\lfloor\log(k+1)\rfloor \le \log 2k = 1+\log k$ on the exponent on the left side while on the right side we can use $k < 1+\lfloor\log(k+1)\rfloor+k$ on the base:

$$\begin{aligned}\Leftarrow (2k)^{\log k+k} &< k^{2k-4}\\ \Leftrightarrow (\log k+k)\log 2k &< (2k-4)\log k\\ \Leftrightarrow (\log k+k)(1+\log k) &< 2k\log k - 4\log k\\ \Leftrightarrow \log k + k + \log^2 k + k\log k &< 2k\log k - 4\log k\\ \Leftrightarrow k+\log^2 k + 5\log k &< k\log k.\end{aligned}$$

The latter inequality is true for all $k \ge 12$ which confirms our claim that for $k = \ell$ the running time of the dynamic program is still asymptotically better than the one of the naive approach.

A linear or even greater number of small gaps compared to local maxima. We now see what happens with the running times if $\ell = ck$ for some $1 < c$, that is, we have running times of

$$T_n(k,l) = ((1+c)k)^2 2^k (1+\lfloor\log(k+1)\rfloor+ck)^{(1+c)k-1}$$

for the naive approach and

$$\begin{aligned}T_d(k,l) &= \big(1 + \lfloor \log(k+1)\rfloor + ck\big)^3 2^{\min(1+\lfloor\log(k+1)\rfloor+ck,k)}((1+c)k)^{1+\lfloor\log(k+1)\rfloor+ck}\\ &= \big(1 + \lfloor \log(k+1)\rfloor + ck\big)^3 2^k((1+c)k)^{1+\lfloor\log(k+1)\rfloor+ck}\end{aligned}$$

for our dynamic program. For this situation we claim that the dynamic program is still better:

$$\begin{aligned}&\big(1 + \lfloor \log(k+1)\rfloor + ck\big)^3 2^k((1+c)k)^{1+\lfloor\log(k+1)\rfloor+ck}\\ &\qquad\qquad < ((1+c)k)^2 2^k\big(1 + \lfloor \log(k+1)\rfloor + ck\big)^{(1+c)k-1}\\ \Leftrightarrow\ &((1+c)k)^{\lfloor\log(k+1)\rfloor+ck-1} < \big(1 + \lfloor \log(k+1)\rfloor + ck\big)^{(1+c)k-4}.\end{aligned}$$

We can now use the upper bound of $\lfloor\log(k+1)\rfloor - 1 \leq \log k$ on the left-hand side and the lower bound of $\log k \leq 1 + \lfloor\log(k+1)\rfloor$ on the right-hand side. With $a = \log c$ the previous inequality holds if the following inequality holds:

$$\begin{aligned}\Leftarrow\ &((1+c)k)^{\log k + ck} < \big(ck + \log k\big)^{(1+c)k-4}\\ \Leftrightarrow\ &\big(\log k + ck\big)\log((1+c)k) < ((1+c)k-4)\log\big(ck+\log k\big)\\ \Leftrightarrow\ &\big(\log k + 2^a k\big)\log\big((1+2^a)k\big) < \big((1+2^a)k-4\big)\log\big(2^a k + \log k\big)\\ \Leftarrow\ &\big(\log k + 2^a k\big)\log\Big(2^{a+1}k\Big) < \big((1+2^a)k-4\big)\log\big(2^a k\big)\\ \Leftrightarrow\ &\big(\log k + 2^a k\big)\big(a+1+\log k\big) < \big(k+2^a k - 4\big)\big(a+\log k\big)\\ \Leftrightarrow\ &a\log k + \log k + \log^2 k + a2^a k + 2^a k + 2^a k\log k\\ &\qquad < ka + k\log k + a2^a k + 2^a k \log k - 4a - 4\log k\\ \Leftrightarrow\ &a\log k + \log k + \log^2 k + 2^a k < ka + k\log k - 4a - 4\log k\\ \Leftrightarrow\ &2^a k + \log^2 k + 4a < ak + (k-a-5)\log k\end{aligned}$$

We can see that for any constant a (and thus constant c) the term on the right-hand grows asymptotically faster than the left-hand side, since the dominating term $k\log k$ grows faster than the dominating term k. We can see that if we set $a = \log\log k$ (which means that $c = \log k$) then the inequality turns into

$$k\log k + \log^2 k + 4\log\log k < k\log\log k + (k-5)\log k - \log k \cdot \log\log k$$

for which we can see that the dominating term on both sides is $k\log k$ and that it holds for all $k > 30$. This means that for $\ell \in o\big(k\log k\big)$ we can guarantee that the running time of the dynamic program is asymptotically faster than the naive approach. Since we have introduced some inequalities to bound both the left and the right side this is probably not the best bound, but it is the best bound we could achieve.

Chapter 9

Conclusion and Open Problems

In this thesis we presented algorithms and hardness results for weak unit disk contact representations as well as results for the one-dimensional colored nearest neighbor graph problem with and without gaps.

9.1 Weak Unit Disk Contact Representations

We have relaxed the notion of UDCRs to weak UDCRs and analyzed their complexity in two different settings. In the first setting the order of the neighbor disks around a disk can be chosen freely when constructing the representation. Here we were able to show a linear algorithm to find a weak UDCR for caterpillars. This included showing that every caterpillar with a weak UDCR also has a grid-restricted weak UDCR. As an upper bound we then showed that it is NP-hard for trees to decide whether they have a weak UDCR. One immediate open question in this setting, is

Open Problem 9.1. *Does the claim in [Bho+21b, Lemma 5] hold? Even further, do lobsters with a weak UDCR always have a grid-restricted weak UDCR?*

Answering these question positively would show that the algorithm by Bhore et al. [Bho+21b] finds a grid-restricted, x-monotone weak UDCR for all lobsters that have a weak UDCR. In their work they also present the following open question which they conjecture to have a positive answer:

Open Problem 9.2 (cf. [Bho+21b, Section 6]). *Can the algorithm presented by Bhore et al. [Bho+21b, Section 5] be extended to find a weak UDCR on the triangular grid in polynomial time for all trees with bounded distance to a backbone?*

If this conjecture were true this would close the gap between the graph classes for which we have an algorithm and those for which we don't. As shown after the NP-hardness proof the problem is in $\exists\mathbb{R}$. If we are only interested in grid-restricted weak UDCRs the problem is also in NP. It is thus natural to pose the following

Open Problem 9.3. *Is the problem of deciding whether a given tree has a weak UDCR in NP? Is the problem $\exists\mathbb{R}$-complete? Is the problem $\exists\mathbb{R}$-complete for more complex graph classes?*

The second setting we analyzed, consists of weak embedded UDCRs which consists of those weak UDCRs where the order of the neighbor disks is fixed and cannot be chosen freely. Here we presented a linear time algorithm for the restricted class of caterpillars with weak embedded UDCRs on the triangular grid with strictly x-monotone backbone disks. Similar to the previous setting one immediate open question is

Open Problem 9.4. *Do strictly x-monotone caterpillars with a weak embedded UDCR also always have a grid-restricted, strictly x-monotone weak embedded UDCR?*

A positive answer to this open problem would show us that our algorithm from Algorithm 4.1 works for any strictly x-monotone caterpillars and not just grid-representable ones. In general, the question is whether there are other (less restrictive) classes for which we can find a polynomial time algorithm:

Open Problem 9.5. *For which other classes of caterpillars with which restrictions on their weak embedded UDCRs can we find polynomial time algorithms?*

We were also able to show that in this setting it is already NP-hard to decide whether caterpillars have a weak embedded UDCR. In addition, the problem is in $\exists\mathbb{R}$ in general and in NP if we are only interested in grid-restricted weak embedded UDCRs. Similar to Open Problem 9.3 we thus natural to pose the following

Open Problem 9.6. *Is the problem of deciding whether a given caterpillar has a weak embedded UDCR in NP? Is the problem $\exists\mathbb{R}$-complete? Is the problem $\exists\mathbb{R}$-complete for more complex graph classes?*

It may also be insightful to look into the embeddings of the given graphs. It is possible that for certain graph classes some properties of embeddings make the problem hard and other properties may make it easy to find a weak embedded UDCR.

9.2 Colored Nearest Neighbor Graphs

We managed to gain structural insight into the one-dimensional colored nearest neighbor graph problem with and without gaps. Our first result was that with just one color we are able to color all input edge sets that exclude exactly the local maxima of the neighbor edges. We then continued with two allowed colors for which we saw that a linear time algorithm is able to solve the problem. For this we observed that it is sufficient to focus our efforts onto a small subset of all possible color assignments, namely the basic color assignments. These observations are independent of the number of colors, however, the linear time algorithm cannot be easily extended to more than two colors. The reason is that for two colors we can use the fact that any problems that may arise, are due to two neighboring special edges. For more colors, any two special edges may be the reason for a conflict. As a first step to solve for more than two colors we studied the number of colors necessary in the worst case. It turned out that the number depends linearly on the number of small gaps as well as logarithmically on the number of local maxima. We finally presented a dynamic programming algorithm that solves the problem with an exponential dependency on the number of colors necessary.

In Section 6.3.3 we have observed that the 1D-CNNG-Gaps problem and the 1D-CNNG problem are both in NP. Looking at the running times of the dynamic program for both problems we can make further observations. Since the running time for the 1D-CNNG problem is $O\left(\log^3 n \cdot 2^{\log^2 n}\right)$ which is quasi-polynomial, it is unlikely that the problem is NP-hard. If no polynomial time algorithm can be found, the problem may be NP-intermediate. We state this as

Open Problem 9.7. *Is there a polynomial time algorithm for the* 1D-CNNG *problem? Is the problem NP-intermediate?*

The running time of the dynamic program for the 1D-CNNG-Gaps problem, on the other hand, is truly exponential. This means that it would not be surprising if it turned out to be NP-complete. We pose this as

Open Problem 9.8. *Is there a polynomial time algorithm for the* 1D-CNNG-Gaps *problem? Is the problem NP-complete?*

We can use Open Problem 9.7 as a stepping stone for this open problem. Additionally, we may try to find easier algorithms for restricted versions of the 1D-CNNG-Gaps problem:

Open Problem 9.9. *Is there a polynomial time algorithm for 1. the* 1D-CNNG-Gaps *problem without any small gaps, or 2. the* 1D-CNNG-Gaps *problem without any local maxima? What is the complexity of those restricted problems?*

Bibliography

[Ala+14] Md. Jawaherul Alam, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, and Sergey Pupyrev. "Balanced Circle Packings for Planar Graphs". In: *Graph Drawing*. GD 2014. Ed. by Christian Duncan and Antonios Symvonis. Vol. 8871. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 125–136. DOI: 10.1007/978-3-662-45803-7_11 (cit. on p. 2).

[Ber+08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry*. Springer, 2008. ISBN: 978-3-540-77973-5 (cit. on p. 77).

[BC87] Sandeep N. Bhatt and Stavros S. Cosmadakis. "The Complexity of Minimizing Wire Lengths in VLSI Layouts". In: *Information Processing Letters* 25.4 (1987), pp. 263–267. ISSN: 0020-0190. DOI: 10.1016/0020-0190(87)90173-6 (cit. on p. 23).

[Bho+21a] Sujoy Bhore, Maarten Löffler, Soeren Nickel, and Martin Nöllenburg. *Unit Disk Representations of Embedded Trees, Outerplanar and Multi-Legged Graphs*. 2021. DOI: 10.48550/arXiv.2103.08416. arXiv: 2103.08416 [cs]. Pre-published (cit. on p. 2).

[Bho+21b] Sujoy Bhore, Maarten Löffler, Soeren Nickel, and Martin Nöllenburg. "Unit Disk Representations of Embedded Trees, Outerplanar and Multi-legged Graphs". In: *Graph Drawing and Network Visualization*. GD 2021. Ed. by Helen C. Purchase and Ignaz Rutter. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 304–317. ISBN: 978-3-030-92931-2. DOI: 10.1007/978-3-030-92931-2_22 (cit. on pp. 2, 3, 15, 167).

[Bow+15] Clinton Bowen, Stephane Durocher, Maarten Löffler, Anika Rounds, André Schulz, and Csaba D. Tóth. "Realization of Simply Connected Polygonal Linkages and Recognition of Unit Disk Contact Trees". In: *Graph Drawing and Network Visualization (GD'15)*. Ed. by Emilio Di Giacomo and Anna Lubiw. Vol. 9411. LNCS. Springer International Publishing, 2015, pp. 447–459. DOI: 10.1007/978-3-319-27261-0_37 (cit. on pp. 2, 9, 12, 69).

[BM04] John Boyer and Wendy Myrvold. “On the Cutting Edge: Simplified O(n) Planarity by Edge Addition”. In: *Journal of Graph Algorithms and Applications* 8.3 (2004), pp. 241–273. ISSN: 1526-1719. DOI: 10.7155/jgaa.00091 (cit. on p. 2).

[BK98] Heinz Breu and David G. Kirkpatrick. “Unit Disk Graph Recognition Is NP-hard”. In: *Computational Geometry: Theory and Applications* 9.1–2 (1998), pp. 3–24. DOI: 10.1016/s0925-7721(97)00014-x (cit. on pp. 2, 9).

[CCN19] Man-Kwun Chiu, Jonas Cleve, and Martin Nöllenburg. “Recognizing Embedded Caterpillars with Weak Unit Disk Contact Representations Is NP-hard”. In: *Proceedings of the 35th European Workshop on Computational Geometry (EuroCG)*. Utrecht, Netherlands, 2019. URL: https://web.archive.org/web/20220517212501/http://www.eurocg2019.uu.nl/papers/47.pdf (cit. on pp. 6, 45).

[Cle20] Jonas Cleve. “Weak Unit Disk Contact Representations for Graphs without Embedding”. In: *Proceedings of the 36th European Workshop on Computational Geometry (EuroCG)*. Würzburg, Germany, 2020. URL: https://web.archive.org/web/20230324122426/https://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_28.pdf (cit. on pp. 2, 6, 15, 22).

[Cle+22] Jonas Cleve, Nicolas Grelier, Kristin Knorr, Maarten Löffler, Wolfgang Mulzer, and Daniel Perz. “Nearest-Neighbor Decompositions of Drawings”. In: *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*. Ed. by Artur Czumaj and Qin Xin. Vol. 227. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 21:1–21:16. ISBN: 978-3-95977-236-5. DOI: 10.4230/LIPIcs.SWAT.2022.21 (cit. on pp. 4, 81).

[EW96] Peter Eades and Sue Whitesides. “The Logic Engine and the Realization Problem for Nearest Neighbor Graphs”. In: *Theoretical Computer Science* 169.1 (1996), pp. 23–37. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(97)84223-5 (cit. on pp. 23–26, 43).

[FMR06] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. “Trémaux Trees and Planarity”. In: *International Journal of Foundations of Computer Science* 17.05 (2006), pp. 1017–1029. ISSN: 0129-0541. DOI: 10.1142/S0129054106004248 (cit. on p. 2).

[HT74] John Hopcroft and Robert Tarjan. “Efficient Planarity Testing”. In: *Journal of the ACM* 21.4 (1974), pp. 549–568. ISSN: 0004-5411, 1557-735X. DOI: 10.1145/321850.321852 (cit. on p. 2).

[Kap14] Tim van Kapel. “Connect the Closest Dot Puzzles”. M.Sc. thesis. Utrecht University, 2014. URL: https://studenttheses.uu.nl/handle/20.500.12932/17601 (cit. on pp. 4, 81).

[KNP15] Boris Klemz, Martin Nöllenburg, and Roman Prutkin. “Recognizing Weighted Disk Contact Graphs”. In: *Graph Drawing and Network Visualization (GD’15)*. Ed. by Emilio Di Giacomo and Anna Lubiw. Vol. 9411. LNCS. Springer International Publishing, 2015, pp. 433–446. DOI: 10.1007/978-3-319-27261-0_36 (cit. on pp. 2, 9).

[KNP22] Boris Klemz, Martin Nöllenburg, and Roman Prutkin. “Recognizing Weighted and Seeded Disk Graphs”. In: *Journal of Computational Geometry* 13.1 (1 2022), pp. 327–376. ISSN: 1920-180X. DOI: 10.20382/jocg.v13i1a13 (cit. on pp. 2, 9, 11).

[KR92] Donald E. Knuth and Arvind Raghunathan. “The Problem of Compatible Representatives”. In: *SIAM Journal on Discrete Mathematics* 5.3 (1992), pp. 422–427. ISSN: 1095-7146. DOI: 10.1137/0405033 (cit. on p. 55).

[Koe36] Paul Koebe. “Kontaktprobleme der konformen Abbildung”. In: *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physikalische Klasse* 88 (1936), pp. 141–164 (cit. on p. 1).

[Lic82] David Lichtenstein. “Planar Formulae and Their Uses”. In: *SIAM Journal on Computing* 11.2 (1982), pp. 329–343. ISSN: 1095-7111. DOI: 10.1137/0211025 (cit. on p. 55).

[Löf+14] Maarten Löffler, Mira Kaiser, Tim van Kapel, Gerwin Klappe, Marc van Kreveld, and Frank Staals. “The Connect-The-Dots Family of Puzzles: Design and Automatic Generation”. In: *ACM Transactions on Graphics* 33.4 (2014), 72:1–72:10. ISSN: 0730-0301. DOI: 10.1145/2601097.2601224 (cit. on p. 4).

[MM17] Joseph S. B. Mitchell and Wolfgang Mulzer. “Proximity Algorithms”. In: *Handbook of Discrete and Computational Geometry*. Ed. by Jacob E. Goodman, Joseph O’Rourke, and Csaba D. Tóth. 3rd ed. Discrete Mathematics and Its Applications. New York: Chapman and Hall/CRC, 2017, pp. 849–874. ISBN: 978-1-315-11960-1. DOI: 10.1201/9781315119601 (cit. on p. 4).

[Nöl24] Martin Nöllenburg. *Personal Communication*. 2024 (cit. on p. 15).

[PY92] Michael S. Paterson and F. Frances Yao. “On Nearest-Neighbor Graphs”. In: *Automata, Languages and Programming*. Ed. by W. Kuich. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992, pp. 416–426. ISBN: 978-3-540-47278-0. DOI: 10.1007/3-540-55719-9_93 (cit. on p. 4).

[PS93] F.P. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Monographs in Computer Science. Springer New York, 1993. ISBN: 978-0-387-96131-6. URL: https://books.google.de/books?id=gFtvRdUY09UC (cit. on p. 77).

[Sch78] Thomas J. Schaefer. “The Complexity of Satisfiability Problems”. In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing - STOC ’78*. The Tenth Annual ACM Symposium. San Diego, California, United States: ACM Press, 1978, pp. 216–226. DOI: 10.1145/800133.804350 (cit. on p. 22).

[She15] Sherm. “The Great Dot Mystery”. In: *Newark Evening Star and Newark Advertiser. Junior Evening Star* (Dec. 3, 1915). ISSN: 2766-5615. URL: https://chroniclingamerica.loc.gov/lccn/sn91064011/1915-12-03/ed-1/seq-21/ (cit. on pp. 81, 82).

[Syr01] Apostolos Syropoulos. “Mathematics of Multisets”. In: *Multiset Processing*. Ed. by Cristian S. Calude, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Red. by Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen. Vol. 2235. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 347–358. ISBN: 978-3-540-45523-3. DOI: 10.1007/3-540-45523-X_17 (cit. on pp. 81–83).

[Ter24] Soeren Terziadis. “Algorithms for Schematic Representations”. PhD thesis. Technische Universität Wien, 2024. DOI: 10.34726/hss.2024.120711 (cit. on p. 15).

[Tip16] Simon Tippenhauer. “On Planar 3-SAT and Its Variants”. M.Sc. thesis. Berlin: Freie Universität Berlin, 2016. URL: http://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Tippenhauer16.pdf (cit. on p. 55).

[Wik23] Wikipedia. *Connect the Dots*. In: *Wikipedia*. Wikimedia Foundation, Inc., 2023. URL: https://en.wikipedia.org/w/index.php?title=Connect_the_dots&oldid=1181114464 (cit. on p. 81).

Appendix A: Supplementary Material

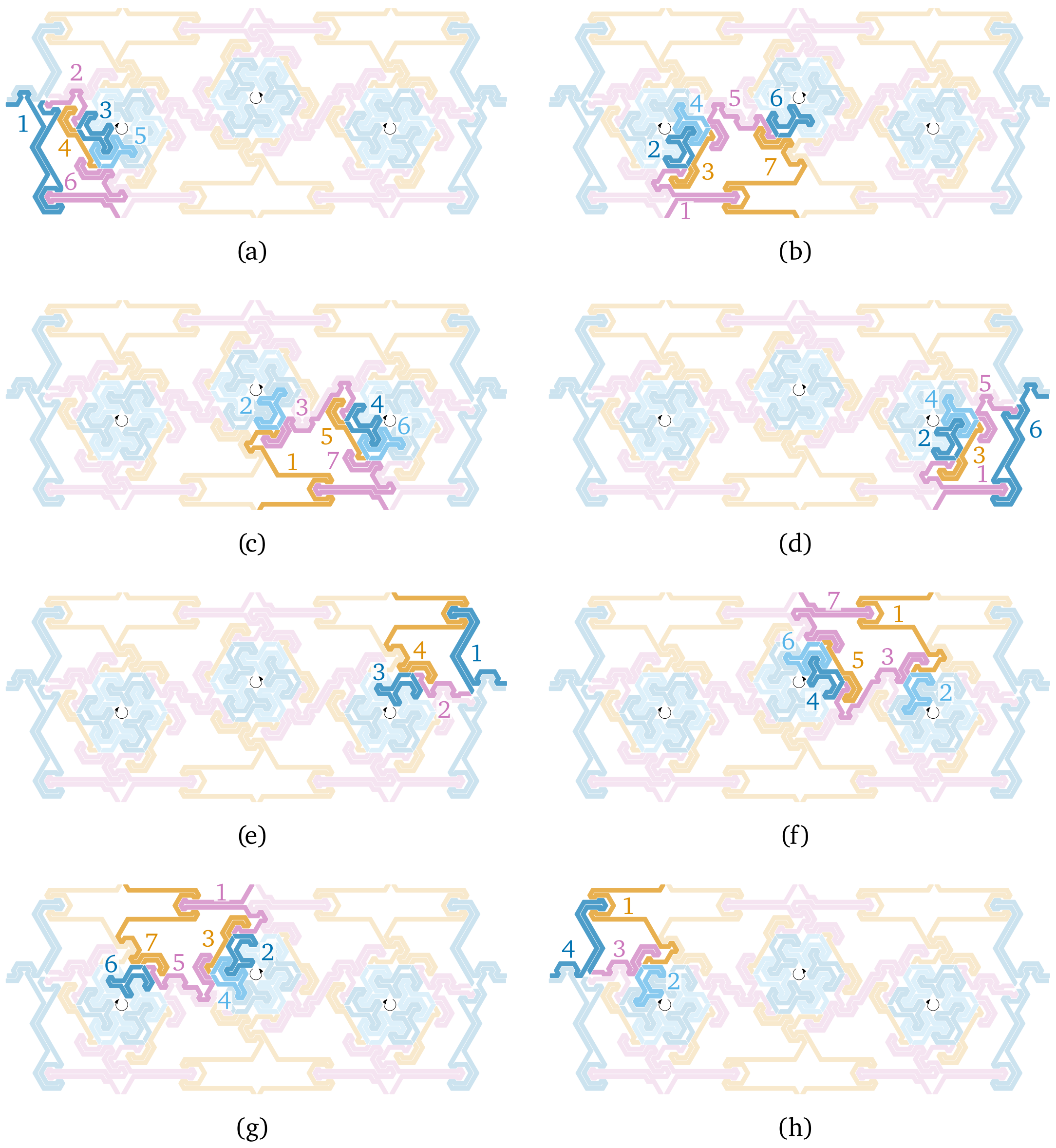


Figure A.1: A breakdown of Figure 4.20b into the parts between outside connectors.

```python
from itertools import chain, combinations, pairwise, permutations, product

def rename(sigma, colors):
    return frozenset(c for i, c in enumerate(colors, start=1) if i in sigma)

def print_combinations(number_of_colors=2, use_all=False):
    colors = tuple(range(1, number_of_colors+1)) # (1, 2, ..., number_of_colors)
    point = list(map(frozenset, chain(combinations(colors, 1), combinations(colors, 2))))

    seen = set()
    for sigma in product(point, repeat=4):
        if sigma in seen:
            continue # ignore assignments that were already seen

        # add sigma and its color renamed and mirrored versions to seen color assignments
        for s in (sigma, tuple(reversed(sigma))):
            for new_colors in permutations(colors):
                seen.add(tuple(map(lambda sig: rename(sig, new_colors), s)))

        if any(sum(map(lambda s: c in s, sigma)) == 1 for c in colors):
            continue # (1) A color c is given to exactly one of the four points.

        ignore = False
        for p1, p2 in pairwise(sigma):
            if not p1 & p2:
                ignore = True
        if ignore:
            continue # (2) Two adjacent points do not have a common color.

        if sigma[0] - sigma[1] or sigma[3] - sigma[2]:
            continue # (3) An outer point has more colors than their adjacent inner point.

        if number_of_colors == 2 and sum(map(len, sigma)) == 8:
            continue # (4) Each point is given both colors (if number of colors is 2).

        if use_all and any(not any(c in s for s in sigma) for c in colors):
            continue # (*) Does not use all colors.

        print(tuple(map(set, sigma)))

print_combinations(2)
# Results:
# ({1}, {1}, {1}, {1})          -> left
# ({1}, {1}, {1, 2}, {2})       -> middle, top
# ({1}, {1}, {1, 2}, {1, 2})    -> right, bottom
# ({1}, {1, 2}, {1, 2}, {1})    -> middle, bottom
# ({1}, {1, 2}, {1, 2}, {2})    -> right, top
# ({1}, {1, 2}, {1, 2}, {1, 2}) -> right, middle

print_combinations(3, True)
# Results:
# ({1}, {1, 2}, {2, 3}, {3})       -> left
# ({1}, {1, 2}, {2, 3}, {2, 3})    -> middle
# ({1, 2}, {1, 2}, {1, 3}, {1, 3}) -> right
```

Listing A.1: Code to enumerate all combinations of color assignments for a local maximum with two (results of `print_combinations(2)` seen in Figure 6.2) and three colors (results of `print_combinations(3, True)` seen in Figure 6.3).

Zusammenfassung

Schwache Kontaktdarstellungen von Einheitskreisscheiben. Eine schwache Kontaktdarstellung von Einheitskreisscheiben (sKvE) eines Graphen ist eine Menge von sich im Inneren nicht schneidenden Einheitskreisscheiben in der Ebene bei der jede Kreisscheibe einem Knoten des Graphen zugeordnet ist. Dabei müssen sich je zwei Kreisscheiben berühren, also im Rand schneiden, wenn die zugehörigen Knoten mit einer Kante verbunden sind. Die Kreisscheiben von nicht per Kante verbundenen Knoten dürfen sich trotzdem berühren. Wir betrachten zwei Varianten des Problems: Bei Graphen ohne Einbettung ist die Platzierung der benachbarten Kreisscheiben egal. Das Problem ist NP-schwer für Bäume und in linearer Zeit lösbar für Raupengraphen (Bäume, die zu Pfaden werden, wenn alle Blätter entfernt werden). Bei Graphen mit einer festen Einbettung ist die Reihenfolge der Platzierung der benachbarten Kreisscheiben vorgegeben. Hier zeigen wir, dass das Problem bereits NP-schwer für allgemeine Raupengraphen ist. Beschränken wir uns auf sKvE, die auf einem regelmäßigen Dreiecksgitter platziert werden können und bei denen die Kreisscheiben der inneren Knoten streng x-monoton sind, so ist das Problem hier ebenfalls in linearer Zeit lösbar.

Gefärbte Nächste-Nachbarn-Graphen. Hier ist eine Menge eindimensionaler Punkte und eine Liste von Strecken zwischen benachbarten Punkten gegeben, sodass jeder Punkt mindestens eine anliegende Strecke hat. Jedem Punkt soll eine nichtleere Menge von Farben zugewiesen werden. Für jede Farbe erzeugen wir eine Kante zwischen diesem Punkt und dem nächstgelegenen mit derselben Farbe. Eine gültige Farbzuordnung erzeugt für jede Strecke genau eine Kante zwischen denselben Endpunkten; zwei Kanten mit verschiedenen Farben zwischen den gleichen Punkten sind verboten. Wählen wir für die Endpunkte jeder Strecke eine eigene Farbe, so ist dies immer eine gültige Lösung. Daher ist die Frage, wie wenig Farben im besten Fall benötigt werden. Für ein und zwei Farben können wir in linearer Zeit eine gültige Farbzuordnung finden, falls sie existiert. Zwei Unterstrukturen sind entscheidend für die Anzahl der benötigten Farben: Lokale Maxima, also Strecken die länger sind als beide benachbarten Strecken, und schmale Lücken, also fehlende Strecken, bei denen mindestens eine benachbarte Strecke länger ist. Die Anzahl der Farben wächst im schlimmsten Fall logarithmisch in der Anzahl der lokalen Maxima und linear in der Anzahl der kleinen Lücken. Wir beschreiben schließlich ein dynamisches Programm, das bei einer Eingabe und einer Anzahl von Farben eine gültige Farbzuordnung mit dieser Anzahl von Farben findet oder zurückgibt, dass keine gültige Farbzuordnung existiert. Die Laufzeit dieses Algorithmus ist exponentiell in der Anzahl der Farben.