ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Unfair Necklace Splitting: Algorithmic and Hardness Results

Master Thesis

Linus Stalder

August 29, 2024

Advisors: Prof. Dr. Bernd Gärtner, Dr. Patrick Schnider, Simon Weber
Department of Computer Science, ETH Zürich

Abstract

Necklace splitting is a famous problem in combinatorics, where two thieves want to fairly share a stolen necklace using the least number of cuts. The necklace consists of multiple types of beads arranged arbitrarily on a string, and their goal is to share the necklace such that each of them gets half the beads of each type. The well-known Ham-Sandwich Theorem implies that they can always share the necklace using at most n cuts, where n is the number of different types of beads.

In this thesis, we deal with a variant of necklace splitting where the thieves do not want to share the necklace fairly. In particular, assume that the thieves agree to split the necklace such that the first thief gets α_i beads of the i-th type of bead. Hence, the goal is to come up with a strategy to divide the necklace according to a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, where the first thief gets α_i beads of type i. In general, it is not possible to divide the necklace according to α using only n cuts anymore. For that to be possible the necklace needs to satisfy a special condition: If the necklace is n-separable, the α-Ham-Sandwich Theorem (a variant of the Ham-Sandwich Theorem) implies that n cuts suffice to divide the necklace according to any possible α-vector. In fact, for any possible α-vector, the cuts are unique and the set of these n cuts is called the α-cut. The computational problem α-Necklace-Splitting gets as input an n-separable necklace and an α-vector and the goal is to output the unique α-cut.

As the main contribution of this thesis, we present a polynomial time algorithm for α-Necklace-Splitting. In contrast to this result, we show that the condition that the necklace is n-separable is necessary to be efficiently tractable. That is, we prove that in general necklaces, deciding whether an α-cut exists is NP-complete. Our result has implications for hardness results of the computational problem corresponding to the α-Ham-Sandwich Theorem. Namely, it implies that it is unlikely that hardness of α-Ham-Sandwich can be shown by proving hardness of α-Necklace-Splitting. This is contrary to what happens for the classic Ham-Sandwich Theorem, where hardness of the corresponding computational problem is shown by proving hardness of fair necklace splitting in general necklaces.

Acknowledgements

I would like to thank my advisors Patrick Schnider and Simon Weber for their continuous support throughout my work on this thesis. Especially, I am grateful for the many discussions we had over the past six months, which have been a major source of inspiration and contributed significantly to the results of this thesis. In addition, I would like to thank Bernd Gärtner for introducing me to his research group by supervising my Bachelor and Semester Theses. This ultimately led me to specialize in theoretical computer science and to be able to present this Master Thesis. Lastly, I would like to thank my family, my friends and everyone who supported me throughout my studies.

Contents

Chapter 1

Introduction

1.1 Motivation

Consider two thieves who stole a necklace. The necklace consists of multiple beads of different types, arranged arbitrarily on a string. Since the thieves want to divide their loot fairly, they want to devise a strategy to cut the necklace in a way, such that each of the thieves receives exactly half of the beads of each type. A simple, but naive, strategy is to cut off every bead from the necklace and share the beads accordingly. However, each cut through the necklace reduces the value of the necklace, because the string itself also consists of some valuable material. Therefore, they want to minimize the number of cuts they have to make to divide the necklace fairly among them. The well-known Ham-Sandwich Theorem (see Theorem 2.9) can be used to show that the thieves can always divide the necklace using only n cuts, where n is the number of different types occurring in the necklace. See Figure 1.1 for an example.

It is a natural question to ask what happens when the thieves do not want to share the necklace fairly. Perhaps one thief prefers one type of bead more than the other thief. In particular, assume that the first thief likes to have an α_i-fraction of the i-th type of bead and the second thief is happy to receive an $(1-\alpha_i)$-fraction of that type. Therefore, they want to split their necklace according to a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ corresponding to the fraction of each

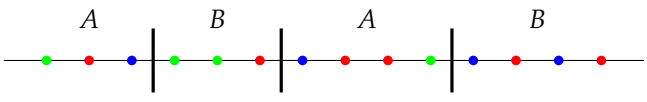


Figure 1.1: A necklace with three types of beads (red, green and blue, or rubies, emeralds and sapphires). Three cuts suffice to split the necklace fairly. The indicated cuts split the necklace into two parts, labelled A and B. Both of these parts contain the same amount of beads per type, so we can give the parts labelled A to the first thief and the parts labelled B to the second thief.

type given to the first thief. Again, they do not want to use too many cuts. Contrary to the case of a fair division, there are necklaces where they need to use more than n cuts. However, if the necklace is such that the different types of beads are not too intertwined, the α-Ham-Sandwich Theorem (see Theorem 2.11) can be used to show that n cuts suffice. The formal condition necklaces need to satisfy to apply this result is called n-separability (see Definition 2.3). A necklace satisfying this condition is called n-separable.

In fact, the α-Ham-Sandwich Theorem implies uniqueness of such splittings for n-separable necklaces for any α-vector. For a given α-vector, we call such a splitting an α-cut. This thesis is motivated by the computational problem of finding an α-cut given an n-separable necklace. The main goal of this thesis is to find an efficient algorithm for this problem or to prove that the problem is hard.

1.2 Outline and Contribution

This thesis is structured as follows. In Chapter 2 we formally define necklaces and formalize the problem described in the previous section; we call this problem α-Necklace-Splitting. The problem gets as input an n-separable necklace with n types of beads and a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ and the goal is to output n cut points dividing the necklace into two parts, such that one part contains an α_i-fraction of the i-th type of beads for all i. The main contribution of this thesis is Chapter 3, where we derive a polynomial time algorithm for α-Necklace-Splitting. Chapter 4 puts this result in the context of computational complexity classes. Moreover, we show that extending α-Necklace-Splitting to arbitrary necklaces (without the condition of n-separability) is NP-hard. That is, the problem of deciding if we can divide the necklace according to an α-vector using only n cuts is shown to be NP-hard. Finally, Chapter 5 highlights some further directions that may be covered by future work on the topic.

During the work of this thesis, we used various computer programs to experiment and verify our results. In particular, we implemented our algorithm for α-Necklace-Splitting. The software developed during this thesis can be accessed at https://github.com/linstald/master-thesis-scripts.

In the remainder of the current chapter, we introduce some notational conventions used throughout this thesis.

1.3 Notation

This thesis uses the following notational conventions.

Natural numbers In this thesis, the *natural numbers* are given by $\mathbb{N} := \{1, 2, \dots\}$, that is, all numbers in $\mathbb{N}$ are strictly positive. Moreover, we define the set $[n]$ for $n \in \mathbb{N}$ to be $[n] := \{1, \dots, n\}$, the natural numbers less than or equal to n.

Graph theory Unless stated otherwise, *graphs* are simple, that is, we do not allow parallel edges or self-loops. If we state that a graph is a *multigraph*, it may contain parallel edges but no self-loops (if we additionally allowed self-loops, we say that it is a *multigraph with self-loops*, but such graphs do not appear in this thesis). For a graph (simple or multigraph) G, its *vertex set* is given by $V(G)$ and its *edges* by $E(G)$. The *degree* of a vertex $v \in V(G)$ is denoted by $\deg(v)$. Moreover, we use $\mu(G)$ to denote the size of a *maximum cut* in G, that is, the maximum number of edges between vertices in A and $V(G) \setminus A$ among all choices of vertices $A \subseteq V(G)$.

For two (multi)graphs G, G' with $V(G) = V(G')$ the *graph sum* $G + G'$ is given by the graph on vertices $V(G)$ and edges $E(G) \cup E(G')$.

A (multi)graph is called *semi-Eulerian* if it contains an *Euler walk*, that is, a sequence of not necessarily distinct vertices $v_1, \dots, v_k \in V(G)$ such that this walk traverses every edge in $E(G)$ exactly once. If the graph contains a closed Euler walk, meaning that $v_1 = v_k$, this walk is called *Euler tour* and the graph is called *Eulerian*.

Geometry In this thesis, we only work in Euclidean geometry. A *hyperplane* in the n-dimensional space $\mathbb{R}^n$ is given by $h := \{x \in \mathbb{R}^n \mid u^\top x = t\}$ for some normal $u \in \mathbb{R}^n$ and distance $t \in \mathbb{R}$. The *positive halfspace* defined by h is given by $h^+ := \{x \in \mathbb{R}^n \mid u^\top x \leq t\}$, and similarly the *negative halfspace* is defined as $h^- := \{x \in \mathbb{R}^n \mid u^\top x \geq t\}$.

A point set $P \subseteq \mathbb{R}^n$ is said to be in *general position* if no $n + 1$ points of P lie on a common hyperplane.

Algorithms and computational complexity We use the *Random-Access-Machine* model to analyse our algorithms, that is, integer operations can be performed in constant time. In this thesis, all algorithms will only use a polynomial amount of memory. Therefore, when deriving a polynomial time algorithm in the Random-Access-Machine model, this implies a polynomial time algorithm in the Turing-Machine model.

Regarding computational *complexity classes*, we will use common complexity classes such as P,NP, coNP, and such without giving formal definitions. For formal definitions of these classes, see for example the book by Arora and Barak [5].

Chapter 2

Necklace Splitting

This chapter serves as an introduction to necklace splitting, the main computational problem under investigation in this thesis. To do so, we introduce necklaces from a combinatorial point of view and list some useful properties of necklaces. Then we discuss the Ham-Sandwich Theorem and a variant of it, called the α-Ham-Sandwich Theorem. We use the latter theorem to introduce the computational problem α-Necklace-Splitting and show that it always has a unique solution. The last section of this chapter refers to related work and discusses similar problems.

2.1 Necklaces and Separability

In this section we formally define necklaces. The definitions are taken and adapted from the work of Borzechowski, Schnider and Weber [15].

An (open) necklace consists of multiple beads of different types arranged sequentially on a string. Of each type, there may be multiple beads in the necklace. We refer to the types as colours. Formally, we can define a necklace as a family of point sets on the number line, where each set of points represents a colour of beads.

Definition 2.1 *A* necklace $C = \{C_1, \ldots, C_n\}$ *with n colours is a family of n pairwise disjoint finite point sets $C_i \subseteq \mathbb{R}$. Each set C_i is called a* colour *and the points $p \in C_i$ are said to be of colour i. The points $c \in C_i$ for some i are called* beads.

This definition is rather geometrical, in the sense that each bead is represented by a fixed point in $\mathbb{R}$. In some cases, it does not matter where the beads are placed exactly, but rather it is relevant where the beads or even the colours are placed relative to each other. Thus, we introduce the necklace string that builds upon that idea.

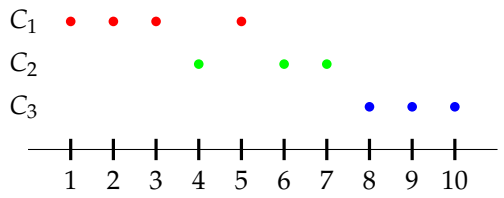


Figure 2.1: A necklace with three colours $C_1 = \{1,2,3,5\}$ (red), $C_2 = \{4,6,7\}$ (green), $C_3 = \{8,9,10\}$ (blue). Its string representation is given by *ababc* (over the alphabet $\{a,b,c\}$). C_3 is an interval, while both C_1 and C_2 are bicomponents. For the two components of C_2 the corresponding subsets are given by $\{4\}$ and $\{6,7\}$ with sizes 1 and 2, respectively.

Definition 2.2 *Let $C = \{C_1, \ldots, C_n\}$ be a necklace. The* necklace string *of C is the string over the alphabet $[n]$ (or some isomorphic set, like the first n letters from the English alphabet, if appropriate), where we order all beads from $-\infty$ to ∞ and place one letter i for consecutive occurring beads from C_i.*

Observe that any two consecutive letters in the necklace string are different. We need the following notation referring to necklaces and necklace strings. Each letter of a necklace string is called a *component*. Thus, for a colour i the number of components of i is given by the number of occurrences of the letter i in the necklace string. A colour with only one component is called *interval*. Colours with exactly two components are called *bicomponents*. Since any component of a colour i corresponds to a subset of C_i, we will use the term component interchangeably for the occurrences of a letter in the necklace string or the corresponding subset. The *size* of a component is given by the number of beads in the subset corresponding to that component. For an example of this notation see Figure 2.1.

In general, the beads on a necklace can be arranged arbitrarily. To quantify how "complex" a necklace is, we introduce the notion of separability. Intuitively, the separability of a necklace measures how much the colours are intertwined.

Definition 2.3 *Let C be a necklace and $k \in \mathbb{N}$. The necklace C is* k-separable *if for all $A \subseteq C$ there exist k separator points $s_1 < \cdots < s_k \in \mathbb{R}$ partitioning $\mathbb{R}$ into intervals $(-\infty, s_1], [s_1, s_2], \ldots, [s_{k-1}, s_k], [s_k, \infty)$ such that when labelling these intervals alternatingly with A^+ and A^- (starting either with A^+ or A^-) we have for all intervals I labelled A^+ that $I \cap \bigcup_{c \in C \setminus A} c = \emptyset$ and for all intervals I' labelled A^- we have $I' \cap \bigcup_{c \in A} c = \emptyset$. We say that $s_1, \ldots, s_k$* separate *the colours A from $C \setminus A$.*

The separability *of C, denoted by $\mathrm{sep}(C)$, is given by the smallest integer $k \geq 0$ such that C is k-separable.*

For an example of the notion of separator points, consider the necklace from Figure 2.1. Let $s_1 = 3.5$, $s_2 = 4.5$ and $s_3 = 5.5$ be three separator points. They partition $\mathbb{R}$ into the intervals $(-\infty, 3.5]$, $[3.5, 4.5]$, $[4.5, 5.5]$ and $[5.5, \infty)$.

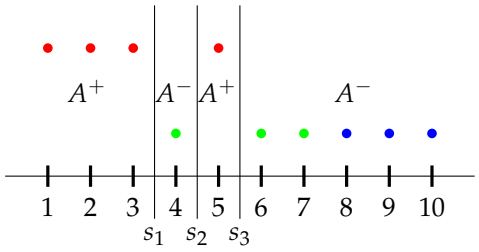


Figure 2.2: An example of three separator points given by $s_1 = 3.5, s_2 = 4.5, s_3 = 5.5$ on the same necklace as in Figure 2.1. When we start labelling the intervals with A^+ all the red beads are contained in the intervals labelled A^+, illustrated by drawing them on top, and the green and blue beads are contained in the intervals labelled A^-, they are drawn on the bottom. Therefore, these separator points separate the colour red from all other colours.

Labelling these alternatingly with A^+ and A^-, starting with A^+, the intervals labelled A^+ are given by $(-\infty, 3.5]$ and $[4.5, 5.5]$, while $[3.5, 4.5]$ and $[5.5, \infty)$ are labelled A^-. When choosing $A = \{C_1\}$ the points s_1, s_2, s_3 are valid separator points according to Definition 2.3, separating A from $C \setminus A$. This is because the intervals labelled A^+ do not intersect with any beads from $C \setminus A = \{C_2, C_3\}$ and the intervals labelled A^- do not intersect with beads from $A = \{C_1\}$. See Figure 2.2 for an illustration of this.

A larger separability means that the necklace can be more complex. However, observe that the separability is always finite, as for any choice of colours $A \subseteq C$, we do not need more separator points than there are beads in the necklace. Also, note that for a necklace C with n colours we have $\text{sep}(C) \geq n - 1$. Indeed, order the colours by their first appearance in the necklace string, that is, C_1 is the colour appearing first in the necklace string, C_2 appears second and so on. When we choose to separate the odd indexed colours from the even indexed colours, there needs to be at least one separator point between the first bead of C_i and the first bead of C_{i+1}.

The separability of the necklace from Figure 2.1 is $\text{sep}(C) = 4$. This is because to separate C_2 from C_1 and C_3 we need to have a separator point in between each transition to or from C_2. Moreover, we do not need more than four separator points to separate any other colour: C_1 can be separated from C_2 and C_3 by the three separator points s_1, s_2, s_3 from above, and C_3 can be separated from C_1 and C_2 by one separator point $s = 7.5$.

In combinatorics, it is often more convenient to use graphs to represent certain structures. This is also the case for necklaces, which can be represented by their walk graph.

Definition 2.4 *The* walk graph *of a necklace $C = \{C_1, \ldots, C_n\}$ is the multigraph $G_C = (V, E)$ where there is a vertex for each colour, $V = [n]$, and for every occurrence of the substring "ab" in the necklace string there is an edge $\{a, b\} \in E$.*

Therefore, the multiplicity of an edge $\{a, b\} \in E$ is given by the number of occurrences of the substrings "ab" and "ba".

By construction, the walk graph is connected. Moreover, traversing the edges as they appear in the necklace from left to right, we visit each edge exactly once. Hence, the walk graph is semi-Eulerian. Also, note that any semi-Eulerian graph is the walk graph of a necklace. This can be seen by traversing an Euler walk in the graph and when traversing a vertex v placing a letter corresponding to v. We obtain a necklace string that has this graph as its walk graph.

The following lemma relates the separability of a necklace to the maximum cut of its walk graph.

Lemma 2.5 ([15]) *Let C be a necklace and G_C its walk graph. The size of a maximum cut of G_C is denoted by $\mu(G_C)$. Then we have $\mu(G_C) = \mathrm{sep}(C)$.*

Proof For a given set $A \subseteq C$ the number of separator points needed to separate A from $C \setminus A$ is given by the number of edges between the sets A and $C \setminus A$. This is because any such edge corresponds to a point where the necklace switches from a colour in A to a colour not in A. Hence, the maximal number of separator points needed to separate any set A from $C \setminus A$ corresponds to the maximum cut $\mu(G_C)$ and vice versa. □

Hence, we can argue about separability by analyzing the maximum cut in the walk graph. The following proposition gives a bound on the maximum cut for multigraphs. It follows from a result by Poljak and Turzík [56], which is a generalization of the Erdős-Edwards bound [28, 29, 31].

Proposition 2.6 ([15]) *A connected multigraph G with n vertices and m edges has a maximum cut $\mu(G)$ of size at least $\omega(G) := \frac{m}{2} + \frac{n-1}{4}$.*

This can now be used to establish an upper bound for the number of edges in the walk graph, depending on the separability of the necklace. Since the separability of a necklace with n colours is at least $n - 1$, let ℓ denote the difference between the separability of the necklace and $n - 1$. Using Proposition 2.6 we can show the following lemma.

Lemma 2.7 *Let C be an $(n - 1 + \ell)$-separable necklace with n colours. Then the number of edges in its walk graph G_C, denoted by m, is at most $m \leq \frac{3n-3}{2} + 2\ell$. In particular, for n-separable necklaces we have $m \leq \frac{3n+1}{2}$.*

Proof By Proposition 2.6, we have $\mu(G_C) \geq \frac{m}{2} + \frac{n-1}{4}$ and from Lemma 2.5

$\mu(G_C) \leq n - 1 + \ell$. Therefore, we have

$$\begin{aligned} & n - 1 + \ell \geq \frac{m}{2} + \frac{n-1}{4} \\ \iff \quad & 4n - 4 + 4\ell \geq 2m + n - 1 \\ \iff \quad & \frac{3n-3}{2} + 2\ell \geq m. \end{aligned}$$

Note that for n-separable necklaces we have $\ell = 1$ and thus $\frac{3n-3}{2} + 2\ell = \frac{3n+1}{2}$. □

Observe that the length of a necklace string is $m + 1$, where m is the number of edges in its walk graph. Thus, Lemma 2.7 shows that for constant ℓ the length of the necklace string is linear in n. Similarly, we can show a lower bound on the number of intervals.

Lemma 2.8 *Let C be a $(n - 1 + \ell)$-separable necklace with n colours. Then the number of intervals, denoted by k, is at least $k \geq \frac{n+1}{2} - 2\ell$. Thus, for n-separable necklaces we have $k \geq \frac{n-3}{2}$.*

Proof Every interval has degree 2 (if it is not at the start or end of the necklace). Moreover, every colour that is not an interval has a degree of at least 4 (if it is not at the start or end of the necklace). We directly get the following bound for the sum of degrees, where we have to subtract 2 for the start and end of the necklace:

$$\begin{aligned} \sum_{c \in C} \deg(c) &\geq k \cdot 2 + (n - k) \cdot 4 - 2 \\ &= 4n - 2k - 2 \end{aligned}$$

In every graph G we have $\sum_{v \in V(G)} \deg(v) = 2 \cdot |E(G)|$ and thus, from Lemma 2.7 we have $\sum_{c \in C} \deg(c) \leq 3n - 3 + 4\ell$. Hence, we must have $k \geq \frac{n+1}{2} - 2\ell$ and when $\ell = 1$ we have $k \geq \frac{n-3}{2}$. □

2.2 The Ham-Sandwich Theorem

Before we define the computational problem of necklace splitting, we take a detour to the Ham-Sandwich Theorem. The Ham-Sandwich Theorem is a famous result that states that a ham sandwich consisting of three ingredients (say, ham, bread and cheese) laying in three-dimensional space can be bisected by a single straight cut. This still works for d ingredients in d dimensions. Originally, this result was proven by Stone and Tukey [63]. However, we use a formulation adapted from Matoušek [50].

Theorem 2.9 (discrete Ham-Sandwich Theorem, [50]) *Let $P_1, \ldots, P_d \subseteq \mathbb{R}^d$ be d finite point sets from $\mathbb{R}^d$. Then there is a hyperplane h such that for all $i \in [d]$*

we have $|h^+ \cap P_i| \geq \lceil |P_i|/2 \rceil$ *and* $|h^- \cap P_i| \geq \lceil |P_i|/2 \rceil$*, where* h^+ *and* h^- *are the two closed halfspaces defined by the hyperplane* h*. If the union of all the point sets are in general position, the inequalities become equalities. The hyperplane* h *is called a* bisecting hyperplane *or* Ham-Sandwich cut*.*

We remark that the standard Ham-Sandwich Theorem works on arbitrary subsets of $\mathbb{R}^d$ using the volume instead of the cardinality of the point sets, however, we only need the discrete version. The classic proof uses the well-known Borsuk-Ulam Theorem [12, 50, 63]. We refer to the corresponding literature for the proofs and further results.

This result already suffices to show that we can divide a necklace fairly using only n cuts [2, 4]. The idea for this is to lift the necklace to the n-dimensional moment curve defined by $f(t) := (t, t^2, \ldots, t^n)$. Then by the Ham-Sandwich Theorem, there is a hyperplane bisecting the lifted points, and the intersections of this hyperplane with the moment curve then directly correspond to the n cutting points. We will use the same idea in the proofs of Lemma 2.12 and Proposition 2.14.

However, we are interested in a more general setting, where we can cut off any arbitrary fraction from any colour. To that end, we need the notion of well-separated point sets.

Definition 2.10 ([15, 62]) *Let* $P_1, \ldots, P_k \subseteq \mathbb{R}^d$ *be finite point sets. They are well-separated if for every choice of indices* $I \subseteq [k]$ *there is a hyperplane separating* $\bigcup_{i \in I} P_i$ *from* $\bigcup_{j \in [k] \setminus I} P_j$*.*

Moreover, we will use a consistent notion for the positive side of a hyperplane. To that end, we follow Steiger and Zhao [62] and Bárány, Hubard and Jerónimo [6] and introduce some notation.

Let $P_1, \ldots, P_d \subseteq \mathbb{R}^d$ be d finite point sets. A transversal hyperplane is a hyperplane defined by points $p_i \in \operatorname{conv}(P_i)$ for all $i \in [d]$ (in general we do not require $p_i \in P_i$). Let $h = \{x \in \mathbb{R}^d \mid u^\top x = t\}$ be a transversal hyperplane defined by the normal vector u and distance t such that $p_i \in h$ for all $i \in [d]$. Note that $-u$ and $-t$ define the same hyperplane. We can make the choice of these unique by choosing the normal u^* such that

$$\det \begin{vmatrix} p_1 & \ldots & p_d & u^* \\ 1 & \ldots & 1 & 0 \end{vmatrix} > 0.$$

Note that any choice of p_i such that $p_i \in h \cap \operatorname{conv}(P_i)$ defines the same hyperplane h. It is not too hard to show that u^* is invariant of this choice of p_i [6]. We can now define the positive halfspace of h by $h^+ := \{x \in \mathbb{R}^d \mid {u^*}^\top x \leq t\}$. The intuition behind this choice is that the d-dimensional simplex formed by the points $p_1, \ldots, p_d, p_1 + u^*$ is oriented positively. For example, in two dimensions this means that the points $p_1, p_2, p_1 + u^*$ form a triangle where these points appear in counterclockwise order.

We will use the following variant of the Ham-Sandwich Theorem.

Theorem 2.11 (α-Ham-Sandwich Theorem, [62]) *For well-separated point sets $P_1, \ldots, P_d$ in general position and any $\alpha = (\alpha_1, \ldots, \alpha_d)$ with $\alpha_i \in \{1, \ldots, |P_i|\}$ there is a unique hyperplane h such that for all $i \in [d]$ we have $h \cap P_i \neq \emptyset$ and $|h^+ \cap P_i| = \alpha_i$. This hyperplane is called the* α-Ham-Sandwich cut.

Observe that the condition $h \cap P_i \neq \emptyset$ implies that the hyperplane is defined by exactly one point of each colour. Hence, h is a transversal hyperplane and h^+ is well-defined, as introduced above. We want to remark that, unlike the traditional Ham-Sandwich Theorem, there is no known proof of Theorem 2.11 using the Borsuk-Ulam theorem.

2.3 α-Cuts and α-Necklace-Splitting

Since we are interested in splitting necklaces, we define α-cuts for necklaces. We will use the α-Ham-Sandwich Theorem to show that such cuts always exist (and are unique) for n-separable necklaces. To do this, we will lift the necklace to the n-dimensional moment curve and apply Theorem 2.11. We need the following lemma.

Lemma 2.12 ([15]) *Let $C = \{C_1, \ldots, C_n\}$ be a necklace. Let C' be the family of point sets in $\mathbb{R}^n$ obtained by lifting each bead of each colour from C to the n-dimensional moment curve defined by $f(t) = (t, t^2, \ldots, t^n)$. Then C' is well-separated if and only if C is n-separable.*

Proof Assume C is n-separable. Consider any set $A \subseteq C$. To separate A from $C \setminus A$ we need at most n separator points $s_1, \ldots, s_n$. Lifting these to the moment curve we obtain n points $s'_1, \ldots, s'_n \in \mathbb{R}^n$. These points define a hyperplane h. At any point s'_i the moment curve passes from one side of h to the other, implying that all points from A are on one side of the hyperplane and all other points on the other side (see [49]). Hence, h is a separating hyperplane for the lifted points in A, implying that C' is well-separated.

Similarly, if C' is well-separated there is a hyperplane separating A from $C' \setminus A$ for all A. This hyperplane intersects the moment curve in at most n points (see [49]), which define n separator points for C. Hence, C is n-separable. □

Similar to the notion of separability, we use separator points to define cuts of necklaces. The idea is that a cut of a necklace is given by a set of separator points and at every separator point the side of the cut changes. Moreover, similar to α-Ham-Sandwich cuts, we require that we cut through each colour exactly once.

Let $C = \{C_1, \ldots, C_n\}$ be a necklace with n colours. A *cut* is a set of n points $S = \{s_1, \ldots, s_n\}$ with $s_1 < \cdots < s_n \in \mathbb{R}$ such that for all $i \in [n]$

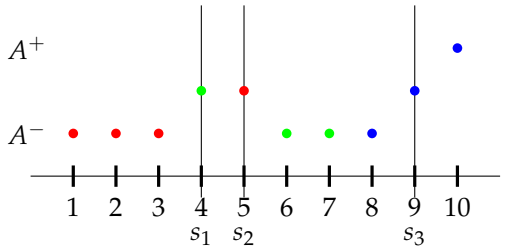


Figure 2.3: An example cut given by the cut points $s_1 = 4, s_2 = 5, s_3 = 9$. Recall that the colours C_1, C_2, C_3 correspond to the red, green and blue beads, respectively. The cut permutation of this cut is $(2,1,3)$, since the first cut point is in C_2, the second in C_1 and the third in C_3. Hence, the cut parity is odd and thus the positive side is given by the intervals starting at an odd indexed separator point. Counting the beads on the positive side (plus the cut beads), we see that this cut is a $(1,1,2)$-cut.

there is a point of colour C_i, that is, $S \cap C_i \neq \emptyset$. To distinguish between separability and cuts, we use the term *cut points* to refer to the points in S. Since all of these points are beads from the necklace, we also use the term *cut beads*. The points partition $\mathbb{R}$ into $n+1$ intervals $(-\infty, s_1], [s_1, s_2], \ldots, [s_n, \infty)$. Define $s_0 := -\infty$ and $s_{n+1} := \infty$. Then the two sides of the cut defined by S are given by the set of intervals starting at an even indexed cut point, $A_0 = \{[s_i, s_{i+1} \mid 0 \leq i \leq n,\ i \text{ even}\}$, and the set of intervals starting at an odd indexed cut point, $A_1 = \{[s_i, s_{i+1} \mid 0 \leq i \leq n,\ i \text{ odd}\}$.

As for α-Ham Sandwich cuts, we want to define a unique positive side, depending on the cut. To do this, we use the *cut permutation*, which is the permutation $\pi : [n] \to [n]$ such that $s_i \in C_{\pi(i)}$. Note that this permutation is well-defined since the cut has exactly one cut point per colour. The permutation π corresponds to the natural order of the colours induced by the cut points, that is, the first entry of π corresponds to the colour of the first cut point and so on. Now the positive side A^+ of the cut S is defined by the parity of π; if π is even define $A^+ := A_0$ and if π is odd define $A^+ := A_1$. The parity of π is called *cut parity*. We also define the negative side A^- of the cut, which is given by $A^- = A_1$ if the cut parity is even and $A^- = A_0$ if the cut parity is odd. Observe that all cut points are on both the positive and the negative side.

With this, we can define α-cuts for necklaces.

Definition 2.13 *Given a necklace $C = \{C_1, \ldots, C_n\}$ and a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$. A cut $S = \{s_1, \ldots, s_n\}$ is an α-cut if for all $i \in [n]$ we have $S \cap C_i \neq \emptyset$ and $\sum_{I \in A^+} |C_i \cap I| = \alpha_i$. That is, there are exactly α_i beads of colour C_i on the positive side of S.*

For an example of an α-cut see Figure 2.3. Note that an α-cut has $|C_i| - \alpha_i + 1$ beads of colour i on its negative side. We use the notation $\overline{\alpha} := (|C_1| - \alpha_1 + 1, \ldots, |C_n| - \alpha_n + 1)$.

We can show uniqueness of all α-cuts for n-separable necklaces.

Proposition 2.14 *Let $C = \{C_1, \ldots, C_n\}$ be an n-separable necklace. For all $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$ there is a unique α-cut.*

Proof As already mentioned, we lift the necklace $C = \{C_1, \ldots, C_n\}$ to the moment curve to obtain the lifted necklace $C' = \{C'_1, \ldots, C'_n\}$ and apply the α-Ham-Sandwich Theorem on C'. Lemma 2.12 implies that C' is well-separated. Also, note that lifting to the moment curve yields a point set in general position since all colours are pairwise disjoint. Thus Theorem 2.11 is applicable. Note that by the same argument as in the proof of Lemma 2.12 any hyperplane obtained from Theorem 2.11 cutting off an α-fraction of C' directly corresponds to a cut of C having an α-fraction of each colour on one side. Note that this implies that the necklace has exactly two candidate cuts for being α-cuts: one by applying Theorem 2.11 with α and one for $\overline{\alpha}$. To show uniqueness, we need the following technical claim.

Claim 2.15 *Let $0 < t_1 < \cdots < t_n$ and $0 < s_1 < \cdots < s_n$ be arbitrary real numbers. Consider the hyperplanes h and g defined by the points lifted to the moment curve respectively (that is, h is defined by the points $t'_1, \ldots, t'_n$ and g by $s'_1, \ldots, s'_n$). Choosing the normals v and u of h and g, respectively, such that*

$$\det \begin{vmatrix} t'_1 & \ldots & t'_n & v \\ 1 & \ldots & 1 & 0 \end{vmatrix} > 0 \quad \wedge \quad \det \begin{vmatrix} s'_1 & \ldots & s'_n & u \\ 1 & \ldots & 1 & 0 \end{vmatrix} > 0$$

implies that $\mathbf{0} = (0, \ldots, 0) \in h^+ \iff \mathbf{0} \in g^+$.

Proof Consider the functions

$$x_1(\lambda) := \lambda t_1 + (1-\lambda)s_1, \quad \ldots, \quad x_n(\lambda) := \lambda t_n + (1-\lambda)s_n.$$

For any $\lambda \in [0,1]$ the points $x_1(\lambda), \ldots, x_n(\lambda)$ mapped to the moment curve – denoted by $x'_1(\lambda), \ldots, x'_n(\lambda)$ – define a hyperplane $H(\lambda)$. Choose its normal $w(\lambda)$ such that $w(0) = v$ and

$$\lambda \mapsto \det \begin{vmatrix} x'_1(\lambda) & \ldots & x'_n(\lambda) & w(\lambda) \\ 1 & \ldots & 1 & 0 \end{vmatrix}$$

is continuous in λ. Note that such a choice exists since the functions x'_i are continuous in λ and thus we can choose the normal $w(\lambda)$ to be continuous in λ. As det is continuous, this choice also makes this determinant continuous in λ. For any λ we have

$$\det \begin{vmatrix} x'_1(\lambda) & \ldots & x'_n(\lambda) & w(\lambda) \\ 1 & \ldots & 1 & 0 \end{vmatrix} > 0$$

as otherwise there is some λ^* where this determinant is zero, by continuity. This is only possible if there are $i \neq j$ with linear dependent $x'_i(\lambda^*)$, $x'_j(\lambda^*)$

– which happens only if $x_i(\lambda^*) = x_j(\lambda^*)$ since we map to the moment curve (see [49]). However, since $t_1 < \cdots < t_n$ and $s_1 < \cdots < s_n$ we have $x_i(\lambda^*) = \lambda^* t_i + (1-\lambda^*)s_i < \lambda^* t_{i+1} + (1-\lambda^*)s_{i+1} = x_{i+1}(\lambda^*)$ for all $i < n$. Hence, all $x_i(\lambda^*)$ are distinct. In particular, this implies that $w(1)$ is pointing in the same direction as u, that is, $w(1) = c \cdot u$ for some positive constant $c \in \mathbb{R}$, as both of these normals have a positive determinant.

Intuitively, this means that points moving along the moment curve without changing their respective location along the curve define a hyperplane that always has the same orientation. In two dimensions this corresponds to the case where two points p_1, p_2 move along the unit parabola where they do not change their respective location, meaning that p_1 is always to the left of p_2 or p_2 is always to the left of p_1. The normal of the line defined by these points always points in the same direction, that is, it always points either towards or against the open side of the parabola.

Moreover, the relation $\mathbf{0} \in H(\lambda)^+$ is invariant under λ. Otherwise, by continuity there is a λ^* such that $\mathbf{0} \in H(\lambda^*)$, however since $0 < t_1 < \cdots < t_n$ and $0 < s_1 < \cdots < s_n$ we have $0 < x_1(\lambda^*) < \cdots < x_n(\lambda^*)$. Therefore, $H(\lambda^*)$ intersects all the points $x'_1(\lambda^*), \ldots, x'_n(\lambda^*) \neq \mathbf{0}$ and $\mathbf{0} \in H(\lambda^*)$ implies that this hyperplane intersects the moment curve in more than n points – a contradiction to the fact that a hyperplane can intersect the moment curve in at most n points.

Observing that $h = H(0)$ and $g = H(1)$ shows the claim. □

Recall that our goal is to show uniqueness of α-cuts for all α in n-separable necklaces. We may assume without loss of generality that for all beads b we have $b > 0$, otherwise shift the whole necklace accordingly; this does not change the structure of the necklace. Now consider the two hyperplanes, h and g, obtained from Theorem 2.11 on the necklace lifted to the moment curve, C'. Assume that h corresponds to α and g to $\overline{\alpha}$, and let u and v be the normals of these hyperplanes. We already argued above that the cuts corresponding to these hyperplanes are the only two candidates for being α-cuts. We now show that exactly one of them is indeed the α-cut.

Since g corresponds to $\overline{\alpha}$, we have an α-fraction of C' in g^-. Let $q'_1 \in C'_1, \ldots, q'_n \in C'_n$ and $r'_1 \in C'_1, \ldots, r'_n \in C'_n$ be the points defining h and g respectively. The cuts in C are then given by the unlifted points $q_1 \in C_1, \ldots, q_n \in C_n$ and $r_1 \in C_1, \ldots, r_n \in C_n$ respectively; these define the cut permutations π and σ. Note that by the definition of the cut permutations, we have $q_{\pi(1)} < \cdots < q_{\pi(n)}$ and similarly $r_{\sigma(1)} < \cdots < r_{\sigma(n)}$. To get uniqueness, we need to show that only one of these cut permutations assigns the side corresponding to h^+ and g^- as the positive side of the α-cut.

By the definition of the normals of the hyperplanes h and g we have

$$\det \begin{vmatrix} q'_1 & \dots & q'_n & u \\ 1 & \dots & 1 & 0 \end{vmatrix} \cdot \det \begin{vmatrix} r'_1 & \dots & r'_n & v \\ 1 & \dots & 1 & 0 \end{vmatrix} > 0$$

In both of these matrices, we can exchange the columns according to π and σ. Since exchanging columns in a matrix multiplies the determinant by -1, and we need to exchange as many columns as there are inversions in the corresponding permutations, the following product

$$\operatorname{sign}(\pi) \det \begin{vmatrix} q'_{\pi(1)} & \dots & q'_{\pi(n)} & u \\ 1 & \dots & 1 & 0 \end{vmatrix} \cdot \operatorname{sign}(\sigma) \det \begin{vmatrix} r'_{\sigma(1)} & \dots & r'_{\sigma(n)} & v \\ 1 & \dots & 1 & 0 \end{vmatrix}$$

is positive. There are two cases, either the permutations have unequal signs and the determinants have unequal signs, or the permutations have equal signs and the determinants have equal signs. In the first case, Claim 2.15 implies that $\mathbf{0} \in h^+ \iff \mathbf{0} \in g^-$. Indeed, if the determinant for u is positive and the determinant for v is negative, we can negate v such that both of the determinants are positive and apply the claim to get $\mathbf{0} \in h^+ \iff \mathbf{0} \in g^-$ (since negating the normal also switches g^+ and g^-). If the determinant for u is negative and positive for v, we negate u and the claim implies $\mathbf{0} \in h^- \iff \mathbf{0} \in g^+$, which is equivalent to $\mathbf{0} \in h^+ \iff \mathbf{0} \in g^-$.

Note that $\mathbf{0}$ in C' corresponds to 0 in C which is to the left of all beads, by assumption. We can therefore use 0 to argue about the cut parity since 0 is on the positive side of the cut if and only if the cut parity is even. Let A^+_π and A^+_σ be the positive sides of the cuts in C induced by the cut permutations π and σ respectively. Since $\operatorname{sign}(\pi) \neq \operatorname{sign}(\sigma)$ the cuts in C have unequal cut parities, which in turn implies that $0 \in A^+_\pi \iff 0 \notin A^+_\sigma$. As both h^+ and g^- contain an α-fraction of C' and $\mathbf{0} \in h^+ \iff \mathbf{0} \in g^-$, only one A^+_π or A^+_σ contains an α-fraction of C. This cut is the α-cut.

Similarly, for the second case where the permutations have equal signs and the determinants have equal signs, Claim 2.15 implies that $\mathbf{0} \in h^+ \iff \mathbf{0} \in g^+$. This is equivalent to $\mathbf{0} \in h^+ \iff \mathbf{0} \notin g^-$, but since the permutations have an equal sign, only one of the cuts has a positive side containing an α-fraction of C. This cut is the α-cut.

Hence, in both cases, there is a unique α-cut, which concludes this proof. □

This gives rise to the following computational problem. Proposition 2.14 shows that a solution always exists.

Definition 2.16 α-Necklace-Splitting

Input:	*An n-separable necklace $C = \{C_1, \dots, C_n\}$,*
	a vector $\alpha = (\alpha_1, \dots, \alpha_n)$, $\alpha_i \in \{1, \dots, \lvert C_i \rvert\}$.
Output:	*The unique α-cut $S = \{s_1, \dots, s_n\}$.*

Since the problem always has a solution and a solution is efficiently verifiable (by constructing the cut permutation and counting the beads on the positive side) α-Necklace-Splitting lies in the complexity class TFNP, which contains total search problems that are verifiable in polynomial time. We refer to Chapter 4 for a more in-depth introduction to complexity classes in TFNP. In Chapter 3, we give a polynomial time algorithm for α-Necklace-Splitting. However, when the condition of n-separability is dropped, the problem may not have a solution anymore. The problem of deciding if there still is a solution in this case is shown to be NP-complete in Chapter 4.

2.4 Related Work

Necklace splitting was introduced by Alon, Goldberg and West [2, 4, 39] as an application of the Borsuk-Ulam Theorem. However, unlike in the setting we introduced above, necklace splitting is introduced by dividing a necklace fairly among two parties. In the notation introduced above, this corresponds to finding an $(\lceil |C_1|/2 \rceil, \ldots, \lceil |C_n|/2 \rceil)$-cut. However, the necklace does not need to be n-separable. In its full generality, one can also address necklace splitting when there are $k \geq 2$ agents among whom the necklace is to be divided fairly [2].

There is a wide range of research covering problems related to necklace splitting. *Consensus halving* is a continuous generalization of necklace splitting, where the necklace is replaced by a set of measures [16, 24, 40, 60, 61]. The area of mass partitions deals with higher dimensional generalizations of consensus halving. The classic problem of finding a Ham-Sandwich cut in the plane [20, 25, 26, 27, 51] constructs the foundation of this area. Other problems include bisecting more masses with more than one line [7, 41, 58, 59].

A more general version of necklace splitting, again in the discrete setting, is given by the *paint shop problem for words* [11, 30, 54]. There, a word over an n-letter alphabet is to be coloured using k colours according to a predefined colouring scheme. That is, for each letter of the alphabet the colouring scheme specifies how many occurrences of that letter within the word have to be coloured with which colour. This has to be done with the fewest possible colour changes. The paint shop problem for words can be viewed as the minimization variant of a generalization of α-Necklace-Splitting where there are k agents to divide the necklace with and no separability guarantee. The colours of the paint shop problem then become the agents and the alphabet of the word become the colours of the necklace. Minimizing the number of colour changes in the paint shop problem corresponds to minimizing the number of cuts to split the necklace.

There are several algorithmic results about necklace splitting and its related

problems. Finding Ham-Sandwich cuts (which then can be turned into necklace splittings) can be done in polynomial time if the dimension (or the number of colours) is fixed [48]. In general, however, finding Ham-Sandwich cuts, necklace splitting and related problems can be located in complexity classes not believed to be tractable efficiently [23, 35, 36, 37]. We discuss some of these results and the computational complexity of α-Necklace-Splitting in Chapter 4. Furthermore, there are algorithms for necklace splitting based upon approximation or additional constraints [3, 46]. Moreover, in a more recent paper by Borzechowski et al. [14] a variant of finding α-Ham-Sandwich cuts is shown to be computationally equivalent to seemingly unrelated problems.

This thesis is motivated by a fixed parameter tractable (FPT) algorithm for necklace splitting by Borzechowski, Schnider and Weber [15], which uses the separability as a parameter. In Chapter 3 we build upon the ideas from this algorithm and extend them to an algorithm for α-Necklace-Splitting.

Chapter 3

Polynomial Time Algorithm for α-Necklace-Splitting

In this chapter, we derive a polynomial time algorithm for α-Necklace-Splitting. The algorithm will be recursive and some parts need an α- and an $\overline{\alpha}$-cut of a smaller necklace to compute the α-cut in the original necklace. To have access to both the α- and $\overline{\alpha}$-cut, we give an algorithm for the following problem.

Definition 3.1 α-$\overline{\alpha}$-Necklace-Splitting

Input:	*An n-separable necklace* $C = \{C_1, \ldots, C_n\}$, *a vector* $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{1, \ldots, \lvert C_i \rvert\}$.
Output:	*A pair* $(S, \overline{S})$, *where* S *is the unique* α*-cut and* $\overline{S}$ *the unique* $\overline{\alpha}$*-cut.*

Overview The algorithm works in three phases: In the first phase, the necklace is reduced to a necklace where each colour consists of at most two components. That is, some components are removed from the necklace such that an α-cut in the obtained necklace can be turned into an α-cut in the original necklace by inserting cuts in the removed components. This is done by two simple observations of the structure of the necklace. In the second phase, the necklace having only colours of at most two components is further reduced, again by leveraging the structure of such necklaces. In the last phase, the necklace is such that we cannot apply any further reduction steps. To solve this problem, we reduce necklace splitting in such necklaces to a colouring problem in the walk graph. This colouring problem is then modelled as an integer linear program, which turns out to be tractable in polynomial time.

3.1 First Phase

In this section we will algorithmically reduce α-$\overline{\alpha}$-Necklace-Splitting to the following subproblem, where each colour in the necklace consists of at most two components.

Definition 3.2 α-$\overline{\alpha}$-Necklace-Splitting$_2$

Input: *An n-separable necklace $C = \{C_1, \ldots, C_n\}$, each colour of at most 2 components, and a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{1, \ldots, |C_i|\}$.*
Output: *A pair $(S, \overline{S})$, where S is the unique α-cut and $\overline{S}$ the unique $\overline{\alpha}$-cut.*

Our reduction is based on the following observation.

Lemma 3.3 *Let C be an n-separable necklace with n colours. Then in the necklace at least one of the following must be true:*

(i) there are two neighbouring intervals, or

(ii) there is no colour with more than four components.

Proof We show that when both (i) and (ii) do not hold we get a contradiction. Let G_C be the walk graph of C.

If there are no two neighbouring intervals, we can take the set of intervals I as a cut. The size of this cut is at least $2|I| - 2$ – each interval has two incident edges, but we need to subtract 2 since if there are intervals at the start and the end of the necklace, these do only have one adjacent edge. Therefore, we have $\mu(G_C) \geq 2|I| - 2$ and thus $|I| \leq \frac{\mu(G_C)+2}{2}$. Since C is n-separable, we have $\mu(G_C) \leq n$ which implies $|I| \leq \frac{n+2}{2}$.

If (ii) does not hold, there is a colour with at least five components. In the walk graph, this colour will have degree 10 (assuming it is not at the start or end of the necklace).

Hence, if both (i) and (ii) do not hold, there are $|I|$ colours with one component, at least one colour with five components and all other colours have at least two components (and thus degree at least 4). Since the first and last components both lack one edge for this calculation, we need to subtract two edges for these. We thus get a lower bound on the sum of degrees in the walk graph.

$$\begin{aligned}\sum_{c \in C} \deg(c) &\geq |I| \cdot 2 + (n - |I| - 1) \cdot 4 + 10 - 2 \\ &= 4n - 2|I| + 4 \\ &\geq 4n - n - 2 + 4 \\ &= 3n + 2\end{aligned}$$

However, by Lemma 2.7 we have that $\sum_{c \in C} \deg(c) \leq 3n + 1$. Therefore, the above result is a contradiction. This implies that there either must be two neighbouring intervals or no colour with more than four components. □

Algorithmically, the conditions (i) and (ii) can be used as follows. If there are two neighbouring intervals, since an α-cut must go through exactly one bead of each colour, there must be a cut in both intervals. Moreover, in between these cuts, there is no other cut bead. Therefore, we remove the two intervals and solve on the newly obtained necklace recursively. Finally, we add the cuts at the right positions in the removed intervals.

If there are no neighbouring intervals, condition (ii) states that all colours have a constant number of components. The strategy will be to look at the colours with more than two components and test out every possible component per colour. That is, for each colour with more than two components we fix a component and remove the other components of that colour. In this smaller necklace (that still consists of n colours) we recursively solve for an α-cut. The necklace for the recursive call will then be a necklace where each colour has at most two components, so we need to solve an α-$\overline{\alpha}$-Necklace-Splitting$_2$ instance in the recursive step.

Since there must be an α-cut, one combination of components must lead to a smaller necklace whose α-cut can be augmented by inserting the cut of each colour in the fixed component. Still, one caveat is that there may be many colours with at least three components, which would make our idea inefficient. The following Lemma (which is a refinement of Lemma 3.3) shows that this is not the case.

Lemma 3.4 *Let C be an n-separable necklace with n colours. In the necklace at least one of the following must be true:*

(i) there are two neighbouring intervals, or

(ii) at most two colours have three or more components.

Proof As before let I be the set of intervals. If there are no neighbouring intervals, we have $|I| \leq \frac{n+2}{2}$, as seen above. Hence, if there are no neighbouring intervals and more than two colours with at least three components, we have

$$\begin{aligned}\sum_{c \in C} \deg(c) &\geq |I| \cdot 2 + 3 \cdot 6 + (n - |I| - 3) \cdot 4 - 2 \\ &= 4n - 2|I| + 4 \\ &\geq 4n - n - 2 + 4 \\ &= 3n + 2\end{aligned}$$

which contradicts Lemma 2.7. □

To show that our recursive calls indeed work, we need to show that removing neighbouring intervals yields a $(n-2)$-separable necklace on $n-2$ colours.

Lemma 3.5 ([15]) *Let C be an n-separable necklace. If C' is a necklace obtained by removing two neighbouring intervals in C, then C' is $(n-2)$-separable.*

Proof We need to show that $\mu(G') \leq n-2$ for the walk graph G' of C'. If we removed the intervals b and c from C then in the walk graph G of C we replaced a path a, b, c, d with an edge $\{a, d\}$ to obtain G'. For a contradiction assume $\mu(G') > n-2$. Let S' be a cut in G' of size $\mu(G')$. We build a cut S in G of size larger than n. S is obtained by S' where b is added if and only if $a \notin S'$ and c is added if and only if $d \notin S'$. This way the edge $\{b, c\}$ is a cut edge in G if and only if $\{a, d\}$ is a cut edge in G'. Moreover, the edges $\{a, b\}$ and $\{c, d\}$ are both cut edges in G, therefore $\mu(G) \geq \mu(G') + 2 > n$, a contradiction that C was n-separable. □

Similarly, we need to show that removing all but one component from a colour does not destroy n-separability. But this is trivially true as only simplifying a necklace cannot increase the separability (also see [15, Lemma 20]).

This lets us conclude the following proposition.

Proposition 3.6 *Let $T_2(n, N)$ be the time to solve α-$\overline{\alpha}$-Necklace-Splitting$_2$ on an n-separable necklace with n colours and a total of N beads. Then α-$\overline{\alpha}$-Necklace-Splitting on an n-separable necklace with a total of N beads and n colours can be solved in at most $\mathcal{O}(T_2(n, N) + n \cdot N)$ time.*

Proof We use Algorithm 1 for α-$\overline{\alpha}$-Necklace-Splitting. The algorithm is recursive and thus, if n is small enough, we can apply a brute force algorithm that iterates through every possible cut and checks if it has found the α-cut or $\overline{\alpha}$-cut. Otherwise, the algorithm proceeds as follows.

In the first step, neighbouring intervals are removed from the necklace and the α- and $\overline{\alpha}$-cut is computed in the obtained necklace recursively. Then in these cuts, the right cut beads are added in the removed intervals to get the α- and $\overline{\alpha}$-cut. We need to make sure to maintain the cut parity when inserting these cuts, so the algorithm checks first if the cut parity switches when inserting these cuts and if yes, it swaps the α- and $\overline{\alpha}$-cuts obtained by the recursive call. That is, if the cut parity switches, the algorithm uses the obtained $\overline{\alpha}$-cut and turns it into the α-cut by inserting the correct cuts in the removed intervals (and similarly turns the α-cut into an $\overline{\alpha}$-cut).

If there are no neighbouring intervals in the necklace, the algorithm determines the set of all colours with at least three components, call this set C_3. If C_3 is empty, the necklace consists only of colours each having at most two components, so we can use an algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ to compute the α- and $\overline{\alpha}$-cut.

Otherwise, if C_3 is not empty, the algorithm iterates over all combinations of components of colours in C_3. That is, each iteration considers a choice $(c_{i_1}, \ldots, c_{i_{|C_3|}})$ of exactly one component c_{i_k} of each colour C_{i_k} in C_3. In this iteration, all components from colours in C_3 except the components from the current choice are removed. We obtain a necklace that still consists of n colours but each colour has at most two components. Therefore, we can use an algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ to find an α- and $\overline{\alpha}$-cut in the new necklace. However, since we removed beads from the colours in C_3 the α might be invalid for these colours. We therefore use a dummy value of 1 for these. Then the algorithm checks, if the obtained cuts can be turned to the corresponding α- or $\overline{\alpha}$-cut by shifting the cuts along the beads within the components of the current iteration.

To show that the algorithm is correct, we need to show that the algorithm returns the correct result in the case when removing neighbouring intervals (line 10), and in the case when removing the components from colours with at least three components (line 29). Moreover, we need to show that the recursive call when removing neighbouring intervals is valid (line 5), and similarly, the calls to the algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ is valid (lines 14 and 21).

By Lemma 3.5 the recursive call when removing neighbouring intervals (line 5) is valid and returns a correct result, as the necklace of the recursive call has $n-2$ colours and is $(n-2)$-separable. Note that inserting cuts in neighbouring intervals either changes the parity of the cut permutation for all cuts or no cut. This can be seen as moving neighbouring intervals as a pair cannot add additional inversions to the cut permutation. Thus, moving them to the beginning of the necklace, there are always either an even or an odd number of inversions, regardless of the cut to augment. We can therefore indeed check whether inserting the cuts in the intervals changes parity and if it does, we swap the α- and $\overline{\alpha}$-cut of the necklace from the recursive call. This way, we make sure that when inserting the correct cut beads in the removed intervals, the obtained cuts are indeed valid α- and $\overline{\alpha}$-cuts. This shows that in this case (line 10) the algorithm indeed returns the unique α- and $\overline{\alpha}$-cuts.

We already argued in the description of the algorithm that the calls to the algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ (lines 14 and 21) are valid since these calls only work on n-separable necklaces that still have n colours and each colour has at most two components. Also, correctness for the case when there are no colours with at least three components (line 14) follows from the correctness of the algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$.

Next, we show that the cut returned for the case when there are colours with at least three components (line 29) is correct. We only argue for the α-cut, the case for the $\overline{\alpha}$-cut is analogous. Notice that in the unique α-cut there is exactly one component per colour in C_3 where the cut bead of that colour

Algorithm 1 FirstPhase

Input $C = \{C_1, \ldots, C_n\}$ an n-separable necklace and $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$.

Returns $S, \overline{S}$ where S is the unique α-cut and $\overline{S}$ the unique $\overline{\alpha}$-cut of C.

if $n \leq 8$ **then**
  return BruteForce(C, α) ▷ checks every possible cut
end if
if there are neighbouring intervals C_i, C_j in C **then**
  $S', \overline{S'} \leftarrow$ FirstPhase$(C \setminus \{C_i, C_j\}, \alpha \setminus \{\alpha_i, \alpha_j\})$
  if inserting cuts in C_i, C_j switches the cut parity **then**
    $S', \overline{S'} \leftarrow \overline{S'}, S$
  end if
  $S, \overline{S} \leftarrow S', \overline{S'}$ with the right $\alpha, \overline{\alpha}$-cuts in C_i, C_j
  return $S, \overline{S}$
end if
▷ There are no neighbouring intervals
$C_3 \leftarrow \{i \in [n] \mid C_i$ has at least three components$\}$
if $|C_3| = 0$ **then**
  return α-$\overline{\alpha}$-Necklace-Splitting$_2(C, \alpha)$ ▷ second phase
end if
$S, \overline{S} \leftarrow \emptyset$
for all combinations $(c_{i_1}, \ldots, c_{i_{|C_3|}})$ of components $c_{i_k} \subseteq C_{i_k}$ for $i_k \in C_3$ **do**
  $C' \leftarrow C \setminus \{C_{i_1}, \ldots, C_{i_{|C_3|}}\} \cup \{c_{i_1}, \ldots, c_{i_{|C_3|}}\}$
    ▷ C' has exactly one fixed component per colour in C_3
    ▷ C' consists only of colours with at most two components
  $\alpha' \leftarrow \alpha$
  $\alpha'_i \leftarrow 1$ for all $i \in C_3$ ▷ Take a dummy value for α'_i
  $S', \overline{S'} \leftarrow \alpha$-$\overline{\alpha}$-Necklace-Splitting$_2(C', \alpha')$ ▷ second phase
  if S' can be shifted to α-cut in C by shifting cuts in $c_{i_1}, \ldots, c_{i_{|C_3|}}$ **then**
    $S \leftarrow S'$ shifted to α-cut in C shifting cuts in $c_{i_1}, \ldots, c_{i_{|C_3|}}$
  end if
  if $\overline{S'}$ can be shifted to $\overline{\alpha}$-cut in C by shifting cuts in $c_{i_1}, \ldots, c_{i_{|C_3|}}$ **then**
    $\overline{S} \leftarrow \overline{S'}$ shifted to $\overline{\alpha}$-cut in C shifting cuts in $c_{i_1}, \ldots, c_{i_{|C_3|}}$
  end if
end for
return $S, \overline{S}$

lies in. Therefore, there must be one iteration (in the for loop in line 17) for exactly that choice of components. For that iteration, the cut obtained by the call of the algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ (in line 21) is a valid α-cut in the necklace C' for all colours except the ones in C_3, where C' is the necklace obtained by removing all components from colours in C_3 except the components from the current choice. Then the cut α-cut in C' can be shifted to an α-cut in C and the algorithm will correctly remember this α-cut (in line 23). Observe that we do not have to worry about changing cut parities, since the necklace C' works on the same set of colours and the cuts are only shifted within components, so the cut permutation does not change. Moreover, since the α-cut is unique the algorithm will only remember one α-cut. Hence, the cut obtained in this case (line 29) will be the correct α-cut.

For the runtime analysis, note that the brute force step takes $\mathcal{O}(N)$ time. Moreover, as long as there are neighbouring intervals the algorithm takes $\mathcal{O}(N)$ per recursive call. In each recursive call the number of colours decreases by 2, so there are at most $\mathcal{O}(n)$ iterations. Hence, the runtime for removing neighbouring intervals is at most $\mathcal{O}(n \cdot N)$.

By Lemma 3.4 we have $|C_3| \leq 2$ and Lemma 3.3 implies that each colour in C_3 has at most four components. Therefore, the number of iterations over components of C_3 (in the loop of line 17) is constant, where each iteration takes time $\mathcal{O}(T_2(n, N) + N)$.

Hence, the total runtime of our algorithm is $\mathcal{O}(n \cdot N + T_2(n, N) + N) = \mathcal{O}(T_2(n, N) + n \cdot N)$. □

Next, we see how the second phase can be implemented in polynomial time.

3.2 Second Phase

The second phase will solve the α-$\overline{\alpha}$-Necklace-Splitting$_2$ problem. More precisely the second phase reduces this problem again to a subproblem. To do this, the algorithm works similarly to the first phase where structures of the necklace that are easy to solve are exploited.

In particular, the second phase works on necklaces with colours of at most two components having at least one of the following structures

1. There are neighbouring intervals, or
2. the first component is an interval, or
3. the last component is an interval, or
4. the first and last components have the same colour.

Note that although we already removed neighbouring intervals in the first phase, the call in line 21 of Algorithm 1 may contain neighbouring intervals

that were introduced by removing some components from the necklace. Hence, the step of removing neighbouring intervals also needs to be done in the second phase.

In the following, we show how the α-cut can be computed for each of the structures 1–4. We already know how to deal with 1. from the first phase.

If the first component is an interval, the idea is to remove that colour and solve recursively on the smaller instance C'. To obtain an α-cut we take the cut in C' and add the right cut in the first component. Note that if the first component is of colour C_i and there are an odd number of colours $C_j \in C'$ such that $j < i$, augmenting the cut will switch the parity of the cut permutation. In this case, adding the first cut in $C_i \notin C'$ adds an odd number of inversions to the permutation. Moreover, since adding a cut in the first component will always flip the positive and the negative side, we have to be careful about what α-vector we solve in C'. Namely, if the first component is such that the cut parity does not flip, we take $\overline{\alpha}$ and if it does, we take α.

The same idea is applied to case 3, where the last component is an interval. We again remove this colour, solve recursively and add a cut in the last component. In this case, adding the cut to the last component does not flip the positive and negative side of the cut, but the permutation flips parity if there are an odd number of colours $C_j \in C'$ such that $j > i$ where the last component is of colour C_i.

However, to apply recursion on C' we need to show that if the first or last component of C is an interval, we may remove it and obtain an $(n-1)$-separable necklace C'. This is easy to see: In the walk graph of C' consider a maximum cut S'. Then consider the cut S which is equal to S' but insert the removed interval on the opposite side of its only neighbour. That is, when v is the removed interval and u is its only neighbour in G, we have $S = S' \cup \{v\}$ if $u \notin S'$ and $S = S'$ if $u \in S'$. This way the edge $\{u, v\}$ is a cut edge and the maximum cut in G is at least $\mu(G') + 1$, implying that $\mu(G') \leq n - 1$.

In the last case, where the first and last components are of the same colour, we use the following idea. Obtain the necklace C' by removing the colour corresponding to the first and last component. Note that C' is $(n-1)$-separable, as any cut in C' can be augmented to a cut in C by placing the colour we removed on the opposite side as a neighbour of that colour in C. This way we gain at least one cut edge and thus the maximum cut of the walk graph of C is strictly larger than the maximum cut of the walk graph of C', implying that C' is $(n-1)$-separable.

Thus, we can apply recursion on C' and get an α- and $\overline{\alpha}$-cut for the remaining colours in C'. We can now find the α-cut in C by first trying if we can augment the α-cut by inserting a cut in the first or last component and if that does not work, trying if we can augment the $\overline{\alpha}$-cut to an α-cut. Similarly,

we can find the $\overline{\alpha}$-cut. This must always be possible, as when considering the α-cut in C, we can remove the cut bead in the colour of the first and last component and get an α- or $\overline{\alpha}$-cut in C'.

Thus, it is open to dealing with necklaces consisting only of colours with at most two components that do not have any of the structures 1–4. Since we do not know how to reduce these necklaces further we define the class of *irreducible necklaces*.

Definition 3.7 *An n-separable necklace C is called* irreducible *if and only if all of the following conditions hold.*

(i) All colours have at most two components,

(ii) there are no neighbouring intervals,

(iii) neither the first nor the last component is an interval,

(iv) the first and the last components are of different colours.

Thus, the second phase reduces α-$\overline{\alpha}$-Necklace-Splitting$_2$ to the following subproblem.

Definition 3.8 α-$\overline{\alpha}$-Irr-Necklace-Splitting

Input: *An irreducible necklace $C = \{C_1, \ldots, C_n\}$,*
a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{1, \ldots, |C_i|\}$.
Output: *A pair $(S, \overline{S})$, where S is the unique α-cut and $\overline{S}$ the unique $\overline{\alpha}$-cut.*

We will see in the third phase how this problem can be tackled. For the sake of completeness, we also give pseudocode for the second phase in Algorithm 2. From the arguments above it follows that it is correct and runs in $\mathcal{O}(T_{irr}(n, N) + n \cdot N)$ time, where $T_{irr}(n, N)$ is the time needed to solve α-$\overline{\alpha}$-Irr-Necklace-Splitting. This proves the following proposition.

Proposition 3.9 *Let $T_{irr}(n, N)$ be the time to solve α-$\overline{\alpha}$-Irr-Necklace-Splitting on an irreducible necklace with n colours and a total of N beads. Then α-$\overline{\alpha}$-Necklace-Splitting$_2$ on an n-separable necklace with a total of N beads and n colours can be solved in at most $\mathcal{O}(T_{irr}(n, N) + n \cdot N)$ time.*

3.3 Third Phase

In the third phase, we show how α-$\overline{\alpha}$-Irr-Necklace-Splitting can be solved in polynomial time. As already mentioned, we will do this by formulating necklace splitting as a colouring problem in the walk graph. This colouring problem will then be formulated as an integer linear program (ILP). While at first glance a reduction to an ILP might be counterintuitive due to the NP-completeness of ILP, it turns out that in our special case, the ILP is tractable in polynomial time.

Algorithm 2 SecondPhase

Input $C = \{C_1, \ldots, C_n\}$ an n-separable necklace where each colour consists of at most two components and $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$.
Returns $S, \overline{S}$ where S is the unique α-cut and $\overline{S}$ the unique $\overline{\alpha}$-cut of C.

1: **if** $n \leq 8$ **then**
2: **return** BruteForce(C, α) ▷ checks every possible cut
3: **end if**
4: **if** there are neighbouring intervals C_i, C_j in C **then**
5: Proceed as in Algorithm 1.
6: **end if**
7: **if** first component is an interval **then**
8: $C_i \leftarrow$ the colour of first component
9: $S', \overline{S'} \leftarrow$ SecondPhase$(C \setminus \{C_i\}, \alpha \setminus \{\alpha_i\})$
10: **if** adding cut in C_i does not flip cut parity **then**
11: $S', \overline{S'} \leftarrow \overline{S'}, S'$
12: **end if**
13: $S, \overline{S} \leftarrow S', \overline{S'}$ with the right $\alpha, \overline{\alpha}$-cut in C_i
14: **return** $S, \overline{S}$
15: **end if**
16: **if** last component is an interval **then**
17: $C_i \leftarrow$ the colour of last component
18: $S', \overline{S'} \leftarrow$ SecondPhase$(C \setminus \{C_i\}, \alpha \setminus \{\alpha_i\})$
19: **if** adding cut in C_i does flip cut parity **then**
20: $S', \overline{S'} \leftarrow \overline{S'}, S'$
21: **end if**
22: $S, \overline{S} \leftarrow S', \overline{S'}$ with the right $\alpha, \overline{\alpha}$-cuts in C_i
23: **return** $S, \overline{S'}$
24: **end if**
25: **if** first and last component, c_1, c_2, are from the same colour C_i **then**
26: $S', \overline{S'} \leftarrow$ SecondPhase$(C \setminus \{C_i\}, \alpha \setminus \{\alpha_i\})$
27: **if** S' can be shifted to α-cut by shifting cut in c_1 or in c_2 **then**
28: $S \leftarrow S'$ shifted to α-cut by shifting cut in c_1 or in c_2
29: **end if**
30: **if** $\overline{S'}$ can be shifted to α-cut by shifting cut in c_1 or in c_2 **then**
31: $S \leftarrow \overline{S'}$ shifted to α-cut by shifting cut in c_1 or in c_2
32: **end if**
33: Find the $\overline{\alpha}$-cut $\overline{S}$ analogous to lines 27–32
34: **return** $S, \overline{S}$
35: **end if**
36: **return** α-$\overline{\alpha}$-Irr-Necklace-Splitting(C, α) ▷ C is irreducible: third phase

This section is structured as follows. First, we analyse the walk graphs of irreducible necklaces. That is, we claim that for a fixed n the walk graphs of all irreducible necklaces with n colours are isomorphic. Next, we show how necklace splitting corresponds to a colouring problem in this walk graph. We then give the formulation of this colouring problem as an ILP. Lastly, we show that the obtained ILP is tractable in polynomial time. To do so, we prove that the treewidth of the primal graph of our ILP is constant.

3.3.1 The structure of the walk graph

In this first part of the treatise of the third phase, we analyse the structure of the walk graph for irreducible necklaces.

First, we claim that the walk graph $G = (V, E)$ of an irreducible necklace with n colours can be characterized by the conditions of the following lemma.

Lemma 3.10 *Let C be an irreducible necklace. Its walk graph $G = (V, E)$ satisfies all of the following conditions.*

a) G is connected,

b) G is semi-Eulerian but not Eulerian,

c) for all vertices $v \in V$ we have $\deg(v) \in \{2, 3, 4\}$,

d) there are no adjacent vertices of degree 2,

e) the maximum cut of G is at most $\mu(G) \leq n$.

Proof Obviously, any walk graph of some necklace needs to be connected. Moreover, since any walk graph of some necklace needs to be semi-Eulerian, so is G. Since the first and last components are of different colours, exactly two colours have odd degree, which implies that G is not Eulerian.

Furthermore, as each colour consists of at most two components the degree of each colour is at most 4. Additionally, since the first and last colours are not intervals, the starting and ending vertex in the necklace string have degree 3 in the walk graph. This implies condition c). Since there are no two neighbouring intervals in an irreducible necklace, there may not be any adjacent vertices of degree 2, hence d) holds as well. Lastly, condition e) just captures that the necklace must be n-separable. □

It is easy to see that any graph satisfying conditions a)–e) corresponds to an irreducible necklace: take an Euler walk in that graph and place a letter corresponding to vertex v whenever traversing v. We get a necklace string where all colours consist of at most two components, no two intervals are neighbouring, the first and the last components are no intervals and the first and last components are of different colours. If a graph G is the walk graph for some irreducible necklace, we call G an *irreducible walk graph*.

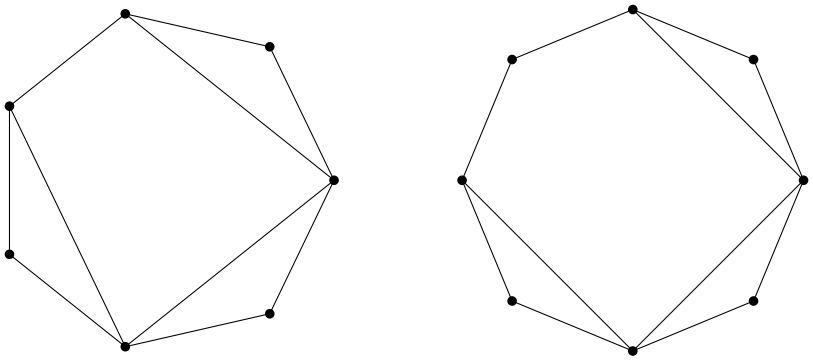


Figure 3.1: The graphs N_7 to the left and N_8 to the right.

We will see that the walk graphs of all irreducible necklaces with n colours are isomorphic. In particular, we claim that the walk graph is isomorphic to the *irreducible necklace graph*, defined as follows.

Definition 3.11 *The cycle graph C_n is the cycle on the vertex set $[n]$. The graph P_{n-1}^{odd} is the graph with vertex set $[n]$ and edge set*

$$E(P_{n-1}^{odd}) = \left\{ \{2i-1, 2i+1\} \mid i \in \left[\lfloor \frac{n-1}{2} \rfloor \right] \right\},$$

that is P_{n-1}^{odd} is the graph on vertex set $[n]$ with a path of length $\lfloor (n-1)/2 \rfloor$ starting at vertex 1 using only the odd vertices. Now the graph N_n obtained by $N_n = C_n + P_{n-1}^{odd}$ is called the irreducible necklace graph *of size n.*

See Figure 3.1 for two examples of these graphs. The third phase heavily relies on the following proposition.

Proposition 3.12 *Let C be an irreducible necklace with n colours for some $n \geq 3$. The walk graph G of C is isomorphic to N_n.*

The proof of this proposition is quite technical and lengthy. Moreover, the proof itself is not instructive for the functionality of the third phase. Thus, we postpone the proof to the end of this chapter.

3.3.2 The cut colouring problem

In this section, we present a colouring problem in the walk graph and show that it is equivalent to necklace splitting. Since the goal of the third phase is to solve α-$\overline{\alpha}$-Irr-Necklace-Splitting, we only focus on irreducible necklaces.

Recall that in the walk graph, there is one vertex per colour. Moreover, traversing the necklace from $-\infty$ to ∞ there is an edge between two colours whenever we switch from one to the other colour. That is, the edges correspond to intervals between two beads – where one of them is the last bead

of one component and the other is the first bead of another component of different colour. Notice that in an α-cut the intervals between beads are completely contained on one side of the cut (positive or negative). Hence, an α-cut induces a colouring of the edges of the walk graph, where each edge is either coloured with a colour associated with the positive side of the α-cut or with another colour associated with the negative side of the α-cut. In the following, we use the colour red to colour edges on the positive side and green for the negative side.

It is worth noting that the colouring is not a two-colouring in the standard notion found in the literature, where each vertex is incident to at most one edge of each colour. In our case there may be multiple incident edges of the same colour per vertex – this would correspond to multiple components that have at least parts on the same side of the cut.

Constraints obtained from the cut Since we are only dealing with irreducible necklaces, each colour consists of at most two components. Hence, every vertex in the walk graph has degree at most 4. Moreover, for every component of a colour c, there are two edges (except if c appears at the beginning or the end of the necklace): one edge corresponding to the change of colours (in the necklace) when entering the component and one edge when leaving the component. We call these pairs of edges *traversals*. For a vertex v define $\mathrm{trav}(v) := \{(e, e') \in E \times E \mid e, e' \text{ is a traversal of } v\}$ to be the set of traversals of v. In Figure 3.3 we will give an example of this notation.

Now note that for a traversal both edges are of the same colour if and only if there is no cut in the corresponding component. Indeed, if there is a cut in a component, the colours of the edges corresponding to the traversal of that component must be of different colours.

As there is exactly one cut per colour and in this section, every colour has at most two components, our colouring must have exactly one red and one green incident edge for an interval and either three red and one green or three green and one red incident edges for bicomponents. This holds for all colours except the first and last one: in these colours, there is no such condition.

The colour graph However, if n is even, the interval from $-\infty$ to the first cut point and the interval from the last cut point to ∞ must be on the same side of the cut, as there are an even number of cut points. Similarly, for n odd, the two intervals are on opposite sides of the cut. To encode this in our colouring, we add an edge between the vertices corresponding to the first and last component if n is even, and a subdivided edge (with a new vertex of degree 2) between these two vertices if n is odd. For the even case, the newly inserted edge must have exactly one colour, encoding the side of the

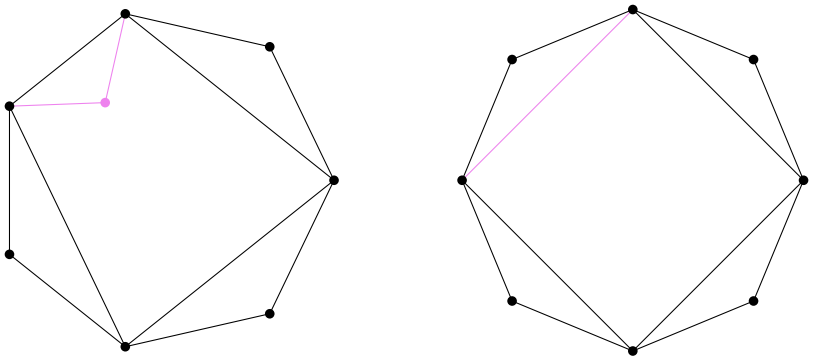


Figure 3.2: Turning the walk graph to the colour graph for $n = 7$ (left) and $n = 8$ (right). The coloured edges and vertices are added to the walk graph to obtain the colour graph.

parts to the left and the right of the necklace. When n is odd, we require the two inserted edges to be of different colours to capture the fact that the two parts are on different sides. Hence, the newly inserted vertex of degree 2 can be treated as a regular interval (of size 0) from the necklace. This newly obtained graph is called the *colour graph*. Therefore, in the colour graph, we can simply enforce the previous conditions on the incident colours on all vertices. See Figure 3.2 for an example of this transformation of the walk graph to the colour graph.

Constraints obtained from α This gives us already a notion of how to translate between a colouring and an α-cut. Note that any α-cut gives a colouring as described above. Moreover, if we are given such a colouring we can get a cut by placing a cut in the component corresponding to the traversal with different coloured edges in each colour of the necklace. However, we do not know in which bead of that component the cut needs to be placed. More importantly, we do not get any information on the underlying α-vector given only a colouring.

To fix this issue, we use the following idea. Consider a colour c with a component c_1 and corresponding traversal (e, e'). Now we assign a weight to the half-edges of (e, e') incident to c equal to the size of c_1. That is, for an edge $e = \{u, c\}$ of that traversal the weight of the half-edge incident to c is $|c_1|$. We denote that with $w_c(e) = |c_1|$. Note that the traversals corresponding to the first and last components also consist of two edges, namely the edge that was there beforehand and the newly inserted edge in the colour graph, used to encode what happens to the left and to the right of the necklace, so there is a weight for every edge.

In this way, we directly get conditions on the α-vector given a colouring, or vice versa, the α-vector imposes conditions on the colouring. For intervals, we do not learn anything new, since we already know that there must be

a cut somewhere in the interval. Therefore, consider a bicomponent c with components c_1 and c_2. If both edges of a traversal are red, we know that the corresponding component must be completely on the positive side of the cut and if both edges are green we know that the component is completely on the negative side. If both edges in c_2 are red, the number of beads on the positive side must be at least $|c_2| + 1$ – we add 1 because the cut bead lies in the c_1 component and the cut bead is also counted on the positive side. Similarly, if both edges in c_2 are green, the number of beads on the positive side must be at most $|c_1|$, because the cut is in the c_1 component and no bead of c_2 is counted on the positive side. These constraints imposed by α are called the α-constraints.

Observe that the colouring obtained by swapping colours of each edge may not satisfy the α-constraints for the same α-vector. In fact, that colouring satisfies the α-constraints for $\overline{\alpha}$. Indeed, for a vertex v with two components c_1 and c_2 if, say, the colouring has different coloured edges for c_1 and two green edges for c_2, we have $\alpha_v \leq c_1$. When swapping the colours, the condition gets $\alpha'_v \geq c_2 + 1$ for some α'_v. We claim that $\alpha'_v = \overline{\alpha}_v$: indeed, $\overline{\alpha}_v = c_1 + c_2 - \alpha_v + 1 \geq c_1 + c_2 - c_1 + 1 = c_2 + 1$. Similarly, this holds when there are two red edges for c_2.

The induced cut Therefore, if we manage to find such a colouring adhering to all these conditions, we learn in which component the cut of each colour needs to be placed. However, knowing for each colour in which component the cut needs to be placed already suffices to get an α-cut. If we know that an α-cut has the cut for a colour c in the component c_1, we can try out every bead in the component of c_1 and check when we have α_c beads on the positive side. We say that this cut is *induced* by the colouring. For an example of such a colouring see Figure 3.3. This figure shows a cut in a necklace together with the corresponding colouring in the colour graph.

Note that for a given colouring, we can swap the colour of each edge and get the same induced cut. Hence, as seen above, there are two colourings satisfying the α-constraints, namely the colouring from the α-cut and the colouring from the $\overline{\alpha}$-cut where the colours of all edges are swapped. However, only one of these colourings induces the α-cut, since only one of them induces a cut where the positive side obtained from the cut parity matches the positive side determined by the red edges from the colouring. Therefore, we require that the positive side of the induced cut matches the side corresponding to the red edges in the colouring. The cut colouring problem is summarized in the following definition.

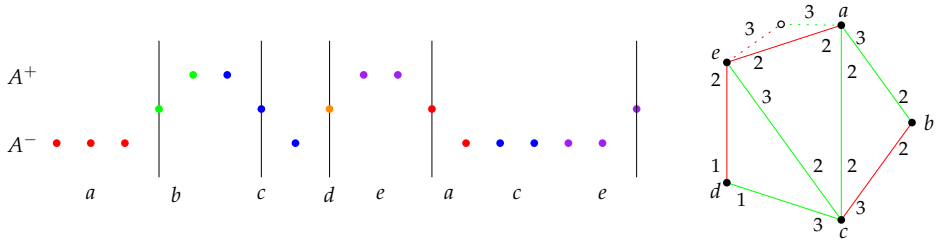


Figure 3.3: A necklace with a necklace string *abcdeace* (left) and its colour graph G (right). The depicted cut is the $(1,2,2,1,3)$-cut, where the colours are ordered as $a < b < c < d < e$. In G the dashed edges together with the unfilled vertex correspond to the subdivision added to the walk graph to obtain the colour graph. The half-edges in G are labelled with the corresponding sizes of the components. For example, consider the vertex c. Its traversals are given by $(\{b,c\},\{c,d\})$ and $(\{a,c\},\{c,e\})$. The half-edges of the first traversal are labelled with a 3, as the first component of c consists of three beads and similarly, the edges of the second traversal are labelled with a 2.

Definition 3.13 α-Cut-Colouring

Input: *The colour graph $G = (V,E)$ of some irreducible necklace $C = \{C_1,\dots,C_n\}$, and a vector $\alpha = (\alpha_1,\dots,\alpha_n)$, $\alpha_i \in \{1,\dots,|C_i|\}$.*

Output: *A colouring of the edges, $\mathcal{R} \subseteq E$, such that (we use $\mathcal{G} := E \setminus \mathcal{R}$):*
(1) $\forall v \in V : |\{e \in \mathcal{R} \mid v \in e\}| \in \{1,3\}$,
(2) $\forall v \in V,\ (e,e') \in \mathrm{trav}(v) : \{e,e'\} \subseteq \mathcal{R} \implies \alpha_v \geq w_v(e) + 1$,
(3) $\forall v \in V,\ (e,e') \in \mathrm{trav}(v) : \{e,e'\} \subseteq \mathcal{G} \implies \alpha_v \leq |C_v| - w_v(e)$,
(4) all edges in $\mathcal{R}$ lie on the positive side of the cut induced by $\mathcal{R}$.

A solution of an α-Cut-Colouring instance for a given α-vector is called *α-colouring*. The concluding claim of this section is the following.

Proposition 3.14 *For an irreducible necklace $C = \{C_1,\dots,C_n\}$ with n colours and a total of N beads, and any vector $\alpha = (\alpha_1,\dots,\alpha_n)$ with $\alpha_i \in \{1,\dots,|C_i|\}$ let $T_{col}(n,N)$ be the time required to solve α-Cut-Colouring on its colour graph. Then we can solve α-$\overline{\alpha}$-Irr-Necklace-Splitting on C in time $\mathcal{O}(T_{col}(n,N)+N)$.*

Proof We use Algorithm 3 to solve α-$\overline{\alpha}$-Necklace-Splitting on the necklace C. The algorithm simply solves α-Cut-Colouring on the colour graph twice using α in one run and $\overline{\alpha}$ in the second run, and translates the colourings back to cuts. In each vertex v, for the component corresponding to a traversal of different colours, we place a cut in that bead such that α_v beads of colour v are on the positive side. Conditions (2) and (3) imply that this is possible for all colours in the necklace and condition (4) makes sure that the obtained cut has the right cut parity and is indeed an α-cut. Also, note that condition (1) ensures that each colour has exactly one cut bead.

We already argued above that an α-cut directly corresponds to an α-colouring. Moreover, any α-colouring can be turned into an α-cut using the induced cut. Observe that this implies that there is a unique α-colouring: If there are two

Algorithm 3 ThirdPhase

Input $C = \{C_1, \ldots, C_n\}$ an n-separable irreducible necklace and $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$.
Returns $S, \overline{S}$ where S is the unique α-cut and $\overline{S}$ the unique $\overline{\alpha}$-cut of C.

```
1: if n ≤ 8 then
2:     return BruteForce(C, α)                ▷ checks every possible cut
3: end if
4: G ← the colour graph of C
5: R_1 ← α-Cut-Colouring(G, α)
6: S ← cut induced by R_1
7: R_2 ← ᾱ-Cut-Colouring(G, ᾱ)
8: S̄ ← cut induced by R_2
9: return S, S̄
```

different α-colourings with different induced cuts, both these cuts are α-cuts, which is not possible as α-cuts are unique, and if there are two different α-colourings with the same induced cut, they can only be the inverse (that is, swapping colours of each edge) of each other. Indeed, when traversing the edges in the order they appear in the necklace, the colours can only change in the components that are cut in the induced cut, implying that the colourings must be the same up to a global swap of colours. But then only one of these colourings satisfies condition (4), implying that there is a unique α-colouring. This implies that the algorithm is correct.

For the runtime, note that the colour graph can be constructed from the necklace in $\mathcal{O}(N)$ time. Moreover, it is easy to see that turning a colouring into a cut can be implemented in $\mathcal{O}(N)$ time. Since we only need to solve α-Cut-Colouring two times, the total runtime is given by $\mathcal{O}(T_{col}(n, N) + N)$.□

3.3.3 Modelling as an integer linear program

In this section, we model α-Cut-Colouring as an integer linear program (ILP). To do this, we only focus on conditions (1)–(3) of an α-Cut-Colouring solution. We formulate an ILP such that any solution of the ILP corresponds to a solution of α-Cut-Colouring satisfying conditions (1)–(3) and vice versa. Condition (4) can then be addressed by observing that for a given instance of α-Cut-Colouring there are at most two solutions satisfying conditions (1)–(3): namely the α-colouring and the colouring obtained by swapping colours of each edge of the $\overline{\alpha}$-colouring, as argued above. In the ILP this then corresponds to the fact that there are only a constant number of feasible solutions, which can be enumerated and checked against condition (4).

Our model will only consist of binary variables. For the sake of clarity, we use **bold** face for variables in our ILP.

To model that each edge is either coloured red or green, we introduce two binary variables per edge. For each edge e in the colour graph add two binary variables $\mathbf{x}_{e,r} \in \{0,1\}$ and $\mathbf{x}_{e,g} \in \{0,1\}$ with the interpretation that $\mathbf{x}_{e,r} = 1 \iff e$ is coloured red and $\mathbf{x}_{e,g} = 1 \iff e$ is coloured green. Since any edge must have exactly one colour we need to add the constraints

$$\mathbf{x}_{e,r} + \mathbf{x}_{e,g} = 1$$

for every edge e.

Next, we model the constraints of the adjacent colours for each vertex. To that end let $e(v,i,1)$ and $e(v,i,2)$ be the two edges corresponding to the i-th traversal of the vertex v. That is $(e(v,i,1), e(v,i,2)) \in \text{trav}(v)$ and in the necklace string this traversal corresponds to the i-th visit of that vertex. In our case $i \in \{1,2\}$ for every vertex; for intervals we always have $i = 1$.

Introduce new binary variables $\mathbf{r}_{v,i}$ and $\mathbf{g}_{v,i}$ for every vertex v and possible indices of traversals i. The interpretation of these variables is that in the i-th traversal of v both the edges are coloured red or green respectively. Thus we would like to add the constraints $\mathbf{r}_{v,i} = \mathbf{x}_{e(v,i,1),r} \cdot \mathbf{x}_{e(v,i,2),r}$ and $\mathbf{g}_{v,i} = \mathbf{x}_{e(v,i,1),g} \cdot \mathbf{x}_{e(v,i,2),g}$. Note that these constraints are not linear, but they can be linearized as follows, by adding the following constraints.

$$\begin{aligned}
\mathbf{r}_{v,i} &\leq \mathbf{x}_{e(v,i,1),r} & \mathbf{g}_{v,i} &\leq \mathbf{x}_{e(v,i,1),g} \\
\mathbf{r}_{v,i} &\leq \mathbf{x}_{e(v,i,2),r} & \mathbf{g}_{v,i} &\leq \mathbf{x}_{e(v,i,2),g} \\
\mathbf{r}_{v,i} &\geq \mathbf{x}_{e(v,i,1),r} + \mathbf{x}_{e(v,i,2),r} - 1 & \mathbf{g}_{v,i} &\geq \mathbf{x}_{e(v,i,1),g} + \mathbf{x}_{e(v,i,2),g} - 1
\end{aligned}$$

We are also interested if the edges corresponding to the i-th traversal have the same colour. We thus introduce the variables $\mathbf{s}_{v,i}$ for each vertex v and its traversals i. They are related to $\mathbf{r}_{v,i}$ and $\mathbf{g}_{v,i}$ as follows.

$$\mathbf{s}_{v,i} = \mathbf{r}_{v,i} + \mathbf{g}_{v,i}$$

Now we use these variables to encode condition (1) of a cut colouring. This can be done by the following constraints.

$$\begin{aligned}
\forall v \in V, \deg(v) = 4 : &\quad \mathbf{s}_{v,1} + \mathbf{s}_{v,2} = 1 \\
\forall v \in V, \deg(v) = 2 : &\quad \mathbf{s}_{v,1} = 0
\end{aligned}$$

Lastly, we need to encode α-constraints, that is, conditions (2) and (3) of a cut colouring. Note that these conditions are only necessary for vertices of degree 4 since for intervals the conditions are always satisfied.

Condition (2) can be implemented using the constraints $\mathbf{r}_{v,i} \cdot w_v(e) + 1 \leq \alpha_v$. For condition (3), we can use the constraints $|C_v| - \mathbf{g}_{v,i} \cdot w_v(e) \geq \alpha_v$. Here,

$|C_v|$ is the total number of beads for that vertex. We obtain the following constraints for all vertices v with $\deg(v) = 4$.

$$\begin{aligned}
\mathbf{r}_{v,1} \cdot w_v(e(v,1,1)) + 1 &\leq \alpha_v \\
\mathbf{r}_{v,2} \cdot w_v(e(v,2,1)) + 1 &\leq \alpha_v \\
|C_v| - \mathbf{g}_{v,1} \cdot w_v(e(v,1,1)) &\geq \alpha_v \\
|C_v| - \mathbf{g}_{v,2} \cdot w_v(e(v,2,1)) &\geq \alpha_v
\end{aligned}$$

Note that in general ILPs also optimize an objective function. However, we are only interested in a satisfying assignment. To summarize our ILP is given as follows.

Definition 3.15 α-Cut-Colouring-ILP

Let $G = (V, E)$ be a colour graph of some irreducible necklace $C = \{C_1, \ldots, C_n\}$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in \{1, \ldots, |C_i|\}$. The α-Cut-Colouring-ILP *is the ILP given by the feasibility of the following constraints*

$$\begin{aligned}
\forall e \in E: \quad & \mathbf{x}_{e,r}, \mathbf{x}_{e,g} \in \{0,1\} \\
\forall v \in V: \quad & \mathbf{r}_{v,i}, \mathbf{g}_{v,i}, \mathbf{s}_{v,i} \in \{0,1\} \\
\forall e \in E: \quad & \mathbf{x}_{e,r} + \mathbf{x}_{e,g} = 1 \\
\forall v \in V: \quad & \mathbf{r}_{v,i} \leq \mathbf{x}_{e(v,i,1),r} \\
& \mathbf{r}_{v,i} \leq \mathbf{x}_{e(v,i,2),r} \\
& \mathbf{x}_{e(v,i,1),r} + \mathbf{x}_{e(v,i,2),r} - 1 \leq \mathbf{r}_{v,i} \\
& \mathbf{g}_{v,i} \leq \mathbf{x}_{e(v,i,1),g} \\
& \mathbf{g}_{v,i} \leq \mathbf{x}_{e(v,i,2),g} \\
& \mathbf{x}_{e(v,i,1),g} + \mathbf{x}_{e(v,i,2),g} - 1 \leq \mathbf{g}_{v,i} \\
& \mathbf{s}_{v,i} = \mathbf{r}_{v,i} + \mathbf{g}_{v,i} \\
\forall v \in V, \deg(v) = 2: \quad & \mathbf{s}_{v,1} = 0 \\
\forall v \in V, \deg(v) = 4: \quad & \mathbf{s}_{v,1} + \mathbf{s}_{v,2} = 1 \\
& \mathbf{r}_{v,1} \cdot w_v(e(v,1,1)) + 1 \leq \alpha_v \\
& \mathbf{r}_{v,2} \cdot w_v(e(v,2,1)) + 1 \leq \alpha_v \\
& |C_v| - \mathbf{g}_{v,1} \cdot w_v(e(v,1,1)) \geq \alpha_v \\
& |C_v| - \mathbf{g}_{v,2} \cdot w_v(e(v,2,1)) \geq \alpha_v
\end{aligned}$$

By construction, α-Cut-Colouring-ILP is equivalent to α-Cut-Colouring (with only the conditions (1)–(3)) in the sense that any solution of the former can be turned to a solution to the latter and vice versa: Simply take the colouring $\mathcal{R} = \{e \in E \mid \mathbf{x}_{e,r} = 1\}$ or the assignment $\mathbf{x}_{e,r} = 1 \iff e \in \mathcal{R}$ and $\mathbf{x}_{e,g} = 1 \iff e \notin \mathcal{R}$.

3.3.4 Solving cut colouring for irreducible necklaces

In this section, we show that for irreducible necklaces α-Cut-Colouring-ILP is tractable in polynomial time. To do that, we leverage the structure of the ILP for irreducible necklaces. We need the following definition.

Definition 3.16 *For an ILP $\mathcal{I}$ given by $\min c^\top \mathbf{x}$ subject to $A\mathbf{x} \leq b$, the primal graph $P(\mathcal{I})$ is the graph having a vertex for each variable $\mathbf{x}_i$ in $\mathcal{I}$ and an edge between two variables $\mathbf{x}_i$ and $\mathbf{x}_j$ if and only if there is a constraint A_k such that $A_{k,i} \neq 0$ and $A_{k,j} \neq 0$. That is, there is an edge if and only if $\mathbf{x}_i$ and $\mathbf{x}_j$ occur together in a constraint.*

We will use an FPT algorithm for ILP having the treewidth of the primal graph as a parameter. The definition of the treewidth of a graph is as follows (adapted from the definition used by Jansen and Kratsch [43]).

Definition 3.17 *Let $G = (V, E)$ be a graph. A tree decomposition $(\mathcal{T} = (\mathcal{V}, A), B : \mathcal{V} \to 2^V)$ consists of a tree $\mathcal{T}$ and a mapping B mapping vertices of $\mathcal{T}$ to subsets of vertices of G, referred to as bags, such that the following three conditions hold.*

1. *for all vertices v of G there is a vertex x of $\mathcal{T}$ with $v \in B(x)$, that is, $\bigcup_{x \in \mathcal{V}} B(x) = V$,*
2. *for all edges $\{u, v\} \in E$ there is a vertex $x \in \mathcal{V}$ such that $\{u, v\} \subseteq B(x)$,*
3. *for all vertices $v \in V$ the induced subgraph $\mathcal{T}[\{x \in \mathcal{V} \mid v \in B(x)\}]$ is connected.*

The width *of a tree decomposition, $\mathsf{wd}(\mathcal{T}, B)$, is then given by the size of a largest bag minus 1. That is, $\mathsf{wd}(\mathcal{T}, B) := \max\{|B(x)| \mid x \in \mathcal{V}\} - 1$. The* treewidth *of G, denoted by $\mathsf{tw}(G)$, is then given by the minimal width among all tree decompositions of G, formally*

$$\mathsf{tw}(G) := \min_{(\mathcal{T},B) \text{ tree decomposition of } G} \mathsf{wd}(\mathcal{T}, B).$$

Our results are based upon the following theorem that directly follows from the results of Jansen and Kratsch [43].

Theorem 3.18 *For an ILP instance $\mathcal{I}$, where each variable is in the domain $\mathcal{D}$, feasibility can be decided in time $\mathcal{O}(|\mathcal{D}|^{\mathcal{O}(\mathsf{tw}(P(\mathcal{I})))} \cdot |\mathcal{I}|)$, where $\mathsf{tw}(P(\mathcal{I}))$ denotes the treewidth of the primal graph of the instance. Moreover, if there are only a constant number of feasible assignments, these can be enumerated in the same time.*

To use this theorem on α-Cut-Colouring-ILP – to show that α-Cut-Colouring-ILP is tractable in polynomial time – we, therefore, have to show that the domain of the variables and the treewidth of the primal graph are bounded. Since the ILP is boolean, the domain is $\mathcal{D} = \{0, 1\}$. Before we bound the treewidth we introduce some notation that will help us later.

Definition 3.19 *Let $G = (V, E)$ be a graph. An* enhanced graph *of G is a graph $\mathcal{G} = (\mathcal{X}, \mathcal{Y})$, with vertices $\mathcal{X} = \{X_1, \ldots, X_k\}$ with $X_i \subseteq V$, where the X_i are called hypervertices, such that $\bigcup_{X_i \in \mathcal{X}} X_i = V$. The edges of $\mathcal{G}$ are given by*

$$\mathcal{Y} = \left\{ \{X_i, X_j\} \subseteq \mathcal{X} \mid X_i \cap X_j \neq \emptyset \text{ or } \exists \{u, v\} \in E : u \in X_i \text{ and } v \in X_j \right\}.$$

An enhanced graph G can be viewed as a graph capturing the structure of G. The vertices of the enhanced graph are subgraphs of G. Hypervertices are connected if they share an element or there is an edge in G connecting an element from one hypervertex and an element from the other. The reason we introduce this notation is the following lemma.

Lemma 3.20 *Consider a graph $G = (V, E)$ with an enhanced graph $\mathcal{G} = (\mathcal{X}, \mathcal{Y})$ of G. Let $w := \max\{|X_i| \mid X_i \in \mathcal{X}\}$ be the size of a largest hypervertex in $\mathcal{G}$. Then the treewidth of G is at most $\mathsf{tw}(G) \leq w \cdot (\mathsf{tw}(\mathcal{G}) + 1) - 1$.*

Proof Let $(\mathcal{T}, B)$ be a tree decomposition of $\mathcal{G}$ of width $\mathsf{tw}(\mathcal{G})$. We now construct a tree decomposition of G of width $w \cdot (\mathsf{tw}(\mathcal{G}) + 1) - 1$. Let $V(\mathcal{T})$ be the vertices of $\mathcal{T}$. Consider the decomposition $(\mathcal{T}, B')$ of G where $B' : V(\mathcal{T}) \to 2^V$ is defined as $B'(x) := \bigcup_{X_i \in B(x)} X_i$ for all $x \in V(\mathcal{T})$. Since $|B(x)| \leq \mathsf{tw}(\mathcal{G}) + 1$ for all $x \in V(\mathcal{T})$ and $|X_i| \leq w$ for all $X_i \in \mathcal{X}$ it follows that the width of this decomposition is at most $w \cdot (\mathsf{tw}(\mathcal{G}) + 1) - 1$. It remains to show that this is a valid tree decomposition of G. To do so, we need to show that the conditions 1–3 from Definition 3.17 are satisfied.

The first two conditions are straightforward. For 1. we have $\bigcup_{x \in V(\mathcal{T})} B'(x) = \bigcup_{x \in V(\mathcal{T})} \bigcup_{X_i \in B(x)} X_i = \bigcup_{X_i \in \mathcal{X}} X_i = V$ using that $(\mathcal{T}, B)$ is a valid tree decomposition of $\mathcal{G}$ and that $\mathcal{G}$ is a graph enhancement of G. Similarly, for 2. consider an edge $e \subseteq X_i$ for some $X_i \in \mathcal{X}$. Then, any $x \in V(\mathcal{T})$ with $X_i \in B(x)$ has $e \subseteq B'(x)$. For an edge $e = \{u, v\} \not\subseteq X_i$ for all $X_i \in \mathcal{X}$, let $X_u, X_v \in \mathcal{X}$ be such that $u \in X_u$ and $v \in X_v$. Then by definition of an enhanced graph, there must be an edge between X_u and X_v in $\mathcal{G}$. Since $(\mathcal{T}, B)$ is a tree decomposition of $\mathcal{G}$ there is an $x \in V(\mathcal{T})$ such that $\{X_u, X_v\} \subseteq B(x)$, implying $\{u, v\} \subseteq B'(x)$.

It is left to prove that for all $v \in V$ the subgraph $\mathcal{T}[\{x \mid v \in B'(x)\}]$ is connected. For that note that $S := \{x \mid v \in B'(x)\} = \{x \mid v \in \bigcup_{X_i \in B(x)} X_i\}$. Use $\mathcal{X}_v := \{X_i \in \mathcal{X} \mid v \in X_i\}$. Then the set S is equal to $S = \{x \mid B(x) \cap \mathcal{X}_v \neq \emptyset\}$. We can rewrite this further as $S = \bigcup_{X_i \in \mathcal{X}_v} \{x \mid X_i \in B(x)\}$.

To show the desired results we need to show that $\mathcal{T}[S]$ is connected. To do so, we show that for any $X_i, X_j \in \mathcal{X}_v$ we have $\{x \mid X_i \in B(x)\} \cap \{x \mid X_j \in B(x)\} \neq \emptyset$. To see why this is sufficient, notice that as $(\mathcal{T}, B)$ is a tree decomposition of $\mathcal{G}$, both $\mathcal{T}[\{x \mid X_i \in B(x)\}]$ and $\mathcal{T}[\{x \mid X_j \in B(x)\}]$ are connected. Thus, if these sets share an element the subgraph induced by the union of these sets must be connected. By induction, this then shows that $\mathcal{T}[S]$ is connected.

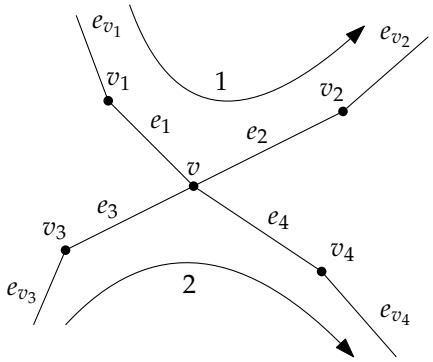


Figure 3.4: A traversal in the colour graph for a vertex v. The arrows indicate the order in which the edges are traversed: in the first traversal e_1 and e_2 and in the second e_3 and e_4.

Note that by definition both X_i and X_j contain v, meaning $X_i \cap X_j \neq \emptyset$. But then, by definition of an enhanced graph, there is an edge between X_i and X_j in $\mathcal{G}$. Further using the definition of a tree decomposition we can conclude that there must be an $x^* \in V(\mathcal{T})$ such that $\{X_i, X_j\} \subseteq B(x^*)$. But then $x^* \in \{x \mid X_i \in B(x)\}$ and $x^* \in \{x \mid X_j \in B(x)\}$. □

Now we are ready to show that the treewidth of the primal graph of the α-Cut-Colouring-ILP is bounded. We show the following proposition.

Proposition 3.21 *Let $C = \{C_1, \ldots, C_n\}$ be an irreducible necklace with n colours. Then the primal graph of the α-Cut-Colouring-ILP of that necklace has treewidth at most 55.*

Proof We proceed in the following steps. First, we show how the primal graph of that ILP looks like. Then we construct an enhanced graph $\mathcal{G}_1$ of that primal graph. In the next step, we analyze $\mathcal{G}_1$ and give an enhanced graph $\mathcal{G}_2$ of $\mathcal{G}_1$, which turns out to be isomorphic to the colour graph. Since we understand the colour graph we can bound its treewidth and we can bound the treewidth of the primal graph by applying Lemma 3.20.

First of all recall that Proposition 3.12 shows how the walk graph of irreducible necklaces looks like. This implies that the colour graph of an irreducible necklace with n colours is always isomorphic to $C_k + C_k^{odd}$, for $k = 2\lceil n/2 \rceil$. Here C_k is a cycle graph with vertices $[k]$ and C_k^{odd} is the cycle graph on the odd vertices in $[k]$.

Now consider the traversals of a vertex v with degree 4 as depicted in Figure 3.4. In the first traversal, we come from an edge e_{v_1} over a vertex v_1 and edge e_1 to v, leaving over edge e_2 through vertex v_2 and edge e_{v_2}. In the second traversal, we come from an edge e_{v_3} going over a vertex v_3 through the edge e_3 to v and leaving to an edge e_4 and vertex v_4 over to an edge e_{v_4}.

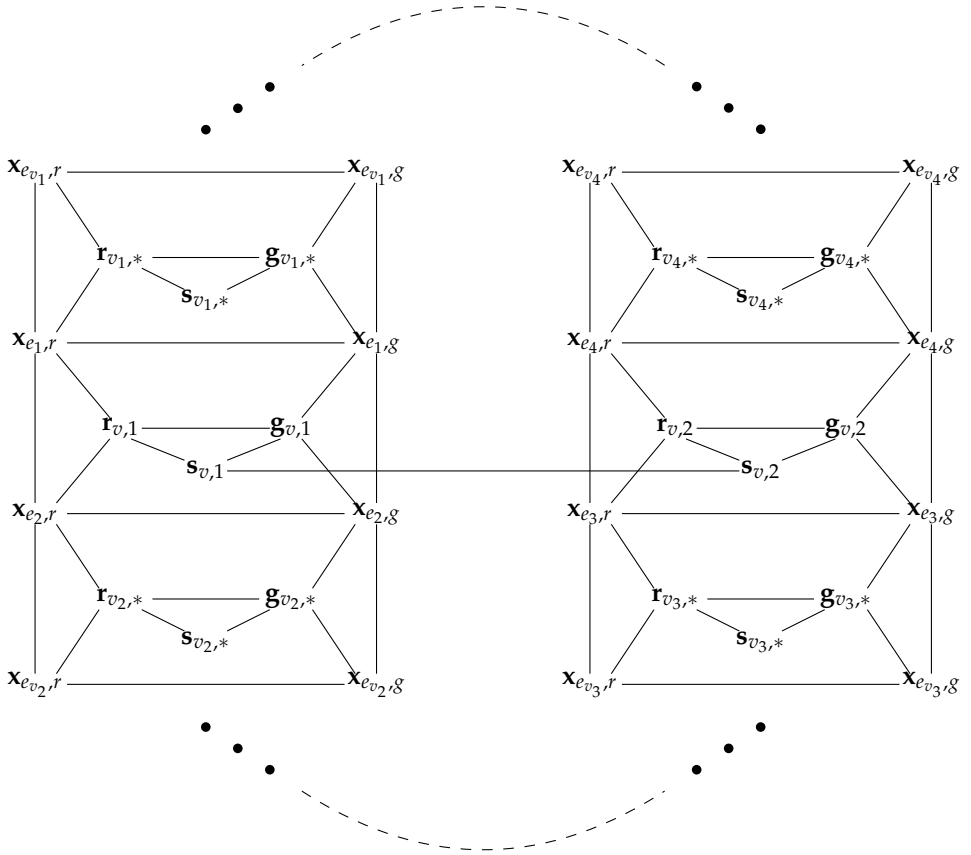


Figure 3.5: The structure of the primal graph of the ILP. The labels correspond to the traversals in Figure 3.4. The *'s indicate that the traversal index for these variables is not relevant for this figure. The dots and dashed lines indicate that the graph continues in a cyclic structure. This cyclic structure corresponds to the Euler tour in the colour graph corresponding to the necklace.

Recall that in the primal graph, we have a vertex for each variable and an edge between variables if there is a constraint using both these variables. Therefore, for an edge e we get the vertices $\mathbf{x}_{e,r}$ and $\mathbf{x}_{e,g}$ connected by an edge, corresponding to the constraint $\mathbf{x}_{e,r} + \mathbf{x}_{e,g} = 1$. Moreover, for a vertex v and i-th traversal e, e' there are vertices $\mathbf{r}_{v,i}, \mathbf{g}_{v,i}, \mathbf{s}_{v,i}$ in the primal graph. We get edges between $\mathbf{x}_{e,r}, \mathbf{x}_{e',r}$ and $\mathbf{r}_{v,i}$ corresponding to the constraint $\mathbf{r}_{v,i} = \mathbf{x}_{e,r} \cdot \mathbf{x}_{e',r}$, similarly we get edges between $\mathbf{x}_{e,g}, \mathbf{x}_{e',g}$ and $\mathbf{g}_{v,i}$. Further there are edges between $\mathbf{r}_{v,i}, \mathbf{g}_{v,i}$ and $\mathbf{s}_{v_i}$ for the constraint $\mathbf{s}_{v,i} = \mathbf{r}_{v,i} + \mathbf{g}_{v,i}$. Lastly, for vertices of degree 4, there is an edge between $\mathbf{s}_{v,1}$ and $\mathbf{s}_{v,2}$ corresponding to the constraint $\mathbf{s}_{v,1} + \mathbf{s}_{v,2} = 1$. All other constraints only include one single variable which does not result in any new edges of the primal graph. Therefore, the primal graph has a structure as shown in Figure 3.5. In particular, the primal graph captures the same cyclic structure as the Euler tour in the colour graph defined by the necklace, where additionally there are chords between the two occurrences of the same vertex.

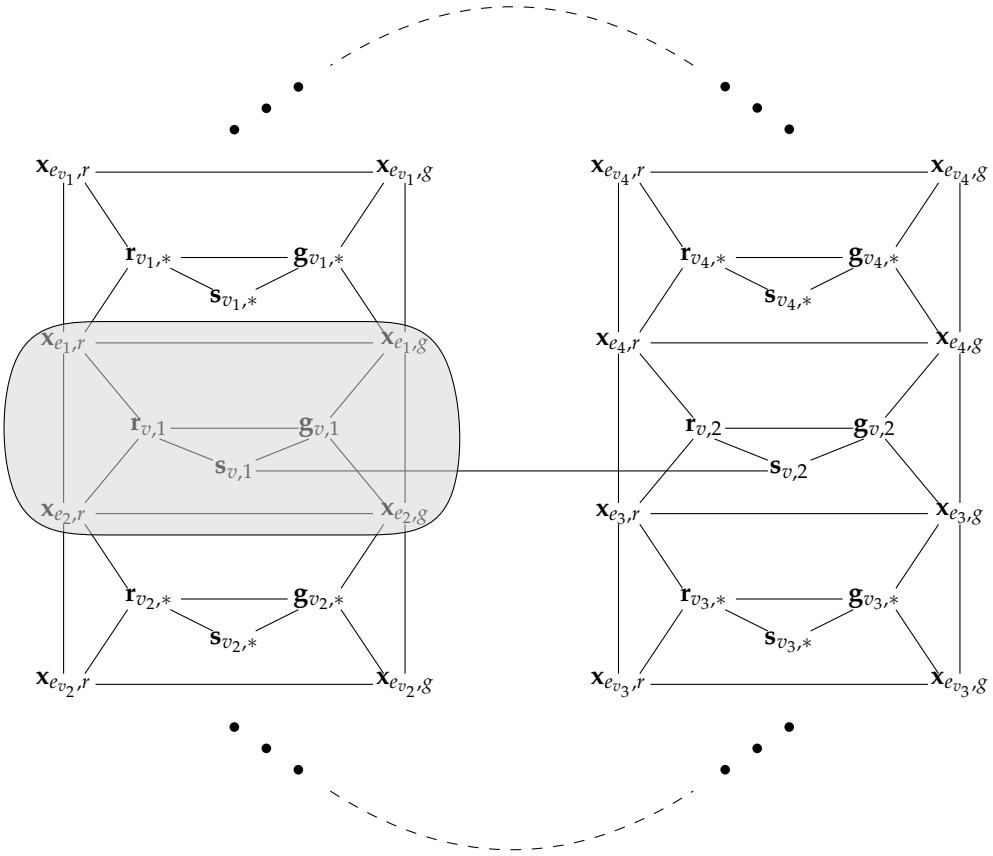


Figure 3.6: The hypervertex for the first traversal of v is given by the vertices in the shaded region. The graph then forms a cycle with chords, where the cycle corresponds to the Euler tour and the chords correspond to the components of the same colour.

As mentioned before, we now construct an enhanced graph of the primal graph. To do this, we define the set of hypervertices as the set of variables occurring in a traversal of a vertex. That is if e, e' are the edges of the i-th traversal of a vertex v, we define $X_{v,i} := \{\mathbf{x}_{e,r}, \mathbf{x}_{e,g}, \mathbf{x}_{e',r}, \mathbf{x}_{e',g}, \mathbf{r}_{v,i}, \mathbf{g}_{v,i}, \mathbf{s}_{v,i}\}$. This is also visualized in Figure 3.6. Therefore, define the enhanced graph $\mathcal{G}_1$ of the primal graph to be the enhanced graph defined by $X_{v,i}$ for all vertices v and traversals i. Note that this is indeed an enhanced graph as any variable occurs in at least one traversal of some vertex.

Now observe that $\mathcal{G}_1$ is the graph corresponding to the Euler tour of the necklace in the sense that for the i-th traversal of a vertex v, there is a vertex $X_{v,i}$ in $\mathcal{G}_1$ and these vertices are adjacent as they appear in the Euler tour. Moreover, for a vertex v of the colour graph with $\deg(v) = 4$, if $X_{v,1}$ and $X_{v,2}$ are the two vertices in $\mathcal{G}_1$ corresponding to the first and the second traversal of v respectively, there is an edge between these vertices. This edge comes from the edge between $\mathbf{s}_{v,1}$ and $\mathbf{s}_{v,2}$ in the primal graph. The structure of $\mathcal{G}_1$

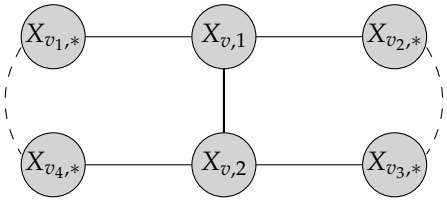


Figure 3.7: The structure of $\mathcal{G}_1$, where the labels correspond to the labels from Figure 3.4 and Figure 3.5. The heavier edge is the one stemming from the edge between $\mathbf{s}_{v,1}$ and $\mathbf{s}_{v,2}$. The dashed line indicates that there is a path between two vertices.

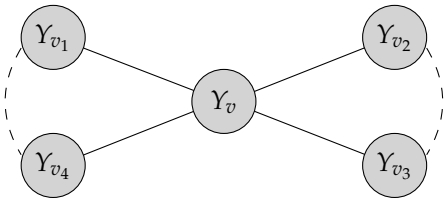


Figure 3.8: The structure of $\mathcal{G}_2$. Note that the only difference to $\mathcal{G}_1$ is that we grouped the edges stemming from $\mathbf{s}_{v,1}$ and $\mathbf{s}_{v,2}$ into a single hypervertex. Then $\mathcal{G}_2$ is isomorphic to the colour graph as in Figure 3.4.

is depicted in Figure 3.7.

This lets us now define an enhanced graph $\mathcal{G}_2$ of $\mathcal{G}_1$ by simply grouping the vertices corresponding to traversals of the same vertex to one hypervertex. That is, for a vertex v we make a hypervertex Y_v containing all vertices $X_{v,i}$ for the traversals i of v. Hence, in $\mathcal{G}_2$ a vertex Y_v has edges to all vertices $Y_{v'}$ where v' is adjacent to v in the colour graph. This implies that $\mathcal{G}_2$ is isomorphic to the colour graph. This is depicted in Figure 3.8.

Now observe that the colour graph – and thus $\mathcal{G}_2$ – has treewidth of at most 3. This can be seen by making a bag for each interval together with its two adjacent vertices. The tree is then given by a path obtained by walking along the cycle of the colour graph. Since the vertex at the start of the cycle will appear in the first and the last bag, we just add it to every bag to get the third property of a tree decomposition. Hence, every bag is of size at most 4.

We have $\mathrm{tw}(\mathcal{G}_2) \leq 3$. By construction, the hypervertices of $\mathcal{G}_2$ contain at most two vertices of $\mathcal{G}_1$. Using Lemma 3.20 we therefore have $\mathrm{tw}(\mathcal{G}_1) \leq 2 \cdot 4 - 1 = 7$. Furthermore, since the hypervertices of $\mathcal{G}_1$ consist of exactly seven vertices of the primal graph, we conclude that the treewidth of the primal graph is at most $7 \cdot 8 - 1 = 55$. □

Using Proposition 3.21 and Theorem 3.18 we can show the following proposition.

Proposition 3.22 *α-$\overline{\alpha}$-Irr-Necklace-Splitting can be solved in $\mathcal{O}(n \cdot N)$ time.*

Proof Theorem 3.18 implies a $\mathcal{O}(n^2)$ algorithm for α-Cut-Colouring-ILP, using that $|\mathcal{D}| = |\{0,1\}| \in \mathcal{O}(1)$ and $\text{tw}(P(\mathcal{I})) \in \mathcal{O}(1)$ by Proposition 3.21. The size of the ILP instance $|\mathcal{I}|$ is given by the number of entries of the matrix defining the instance $\mathcal{I}$. Thus we have $|\mathcal{I}| \in \mathcal{O}(n \cdot n)$, as we have a constant number of variables and constraints per colour. As argued above, the ILP only has a constant number of feasible solutions. Hence, we can enumerate these efficiently, and determine which of these satisfy condition (4) of α-Cut-Colouring. This check can be implemented in $\mathcal{O}(N)$. Hence α-Cut-Colouring can be solved in time $\mathcal{O}(n^2 + N)$. Therefore, by Proposition 3.14 α-$\overline{\alpha}$-Irr-Necklace-Splitting can be solved in $\mathcal{O}(n^2 + N) \in \mathcal{O}(n \cdot N)$ time, using that $n \in \mathcal{O}(N)$. □

Together with the results from the first and second phase in Proposition 3.6 and Propsition 3.9, we can conclude that α-$\overline{\alpha}$-Necklace-Splitting, and thus α-Necklace-Splitting, can be solved in $\mathcal{O}(n \cdot N)$ time. We state this result as a theorem.

Theorem 3.23 *α-Necklace-Splitting on a necklace with n colours and N total beads can be solved in $\mathcal{O}(n \cdot N)$ time.*

3.4 Proving Proposition 3.12

In this last section of this chapter, we prove Proposition 3.12. Recall the statement of said proposition: It claims that for any irreducible necklace C with n-colours, the walk graph of C is always isomorphic to the irreducible necklace graph N_n. Moreover, the graph N_n is defined in Definition 3.11 as $N_n = C_n + P^{odd}_{n-1}$. Recall that C_n is the cycle graph with vertices $[n]$ and P^{odd}_{n-1} is the graph on the vertex set $[n]$ with a path of length $\lfloor (n-1)/2 \rfloor$ on the odd vertices. Also recall the conditions for the walk graphs of irreducible necklaces that we already derived in Lemma 3.10:

Lemma 3.10 *Let C be an n-separable irreducible necklace. Its walk graph $G = (V, E)$ satisfies all of the following conditions.*

a) G is connected,

b) G is semi-Eulerian but not Eulerian,

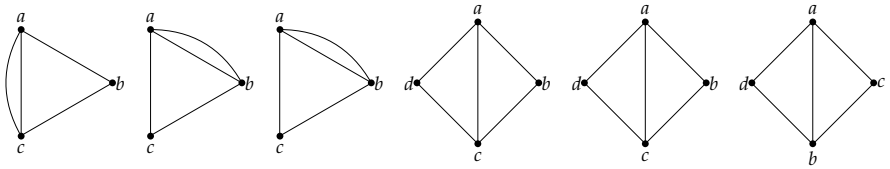
c) for all vertices $v \in V$ we have $\deg(v) \in \{2, 3, 4\}$,

d) there are no adjacent vertices of degree 2,

e) the maximum cut of G is at most $\mu(G) \leq n$.

To show this proposition we distinguish the cases when n is even and when n is odd. We show the result for even n directly and later we will reduce the case for odd n to the even case. It is easy to verify that for $n \leq 4$

the proposition holds, as in this case the only irreducible necklaces (up to permutations) are



all of which have a walk graph isomorphic to N_n. This can be verified by generating all necklaces with $n \leq 4$ and checking if they are irreducible. Lemma 2.7 gives an upper bound on the length of n-separable necklaces, which for $n = 4$ gives a length of at most 8. Hence, there are a total of $4^8 = 65\,536$ possible strings, which can be checked to be irreducible necklace strings with a computer program. We therefore assume $n \geq 5$ throughout the remainder of this section.

3.4.1 Proof for even n

First focus on the case when n is even. Throughout, let G be an irreducible walk graph, that is, a walk graph for some irreducible necklace C with n colours. To prove Proposition 3.12, we proceed as follows. The first and main step is to show that in G we have the correct number of intervals[1]. Having that, we will show that in G we find the same local structures as in N_n. In particular, we will show that bicomponents are adjacent to exactly two intervals and that the adjacent bicomponents of an interval are adjacent. Then, we can prove the graph isomorphism by giving an explicit construction of N_n from G. The main tool in these steps is to derive a contradiction by constructing a cut in G that implies $\mu(G) > n$.

We start by analysing the number of intervals in G. Let I be the set of intervals in G and denote its size with $k = |I|$. First notice that condition d) of irreducible walk graphs states that there are no adjacent intervals. Moreover, by condition c) all intervals have degree 2, and thus the cut given by I has size $2k$. By condition e) we have $2k \leq \mu(G) \leq n$, which implies that $k \leq n/2$. Additionally, Lemma 2.8 directly implies that $k \geq \frac{n-3}{2}$. This shows that for any n (even or odd), we have that $k \in \{\lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor\}$. We summarize this result in a lemma.

Lemma 3.24 *Let G be an irreducible walk graph with n vertices for some $n \geq 5$. Then the number of vertices of degree 2, denoted by k, is $k \in \{\lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor\}$.*

Observe that the irreducible necklace graph N_n has exactly $\lfloor n/2 \rfloor$ intervals. Thus, we need to show that $k = \lfloor n/2 \rfloor$, which is $k = n/2$ for even n. We

[1] In the following we interchangeably use the terms intervals and degree 2 vertices. Similarly, we use bicomponent to refer to vertices of degree at least 3.

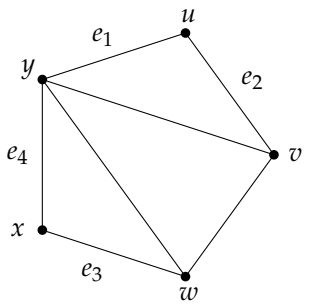


Figure 3.9: For an example of the notion of labelling edges incident to intervals consider the graph as depicted. There are two intervals, u and x. The edges incident to these intervals are $E_I = \{e_1, e_2, e_3, e_4\}$. The labels of these edges are given by $l(e_1) = y$, $l(e_2) = v$, $l(e_3) = w$, $l(e_4) = y$. Hence, the vertices v and w have multiplicity 1 and y has multiplicity 2.

will do this by contradiction, that is, we show that $k = n/2 - 1$ implies that $\mu(G) > n$, which is a contradiction to condition e) of irreducible walk graphs. To do so, we will need a claim that will give us two cases that we can use to distinguish the structure of the walk graph. But first, we need some notation.

Let E_I be the set of edges incident to intervals. For any $e \in E_I$ we label the edge with the bicomponent incident to e. That is, for an edge $e = \{u, v\} \in E_I$ with $\deg(u) = 2$ and $\deg(v) \geq 3$, we label e with $l(e) := v$. This label is well-defined since all edges in E_I are incident to exactly one bicomponent.

Now let $L \subseteq V(G)$ be the multiset of labels of the edges in E_I, that is, each label is in L once for every edge with that label. Formally,

$$L = \{v^{m(v)} \mid v \in V(G),\ \deg(v) \geq 3\}$$

where $m(v) := |\{e \in E_I \mid l(e) = v\}|$ is the multiplicity of the label v. For an example of this notation, see Figure 3.9.

Assuming $k < n/2$, we can apply a counting argument to obtain the following claim.

Claim 3.25 *Assuming $k < n/2$, we have at least one of the following:*

1. *there is at least one vertex $v \notin I$ such that $v \notin L$ – meaning $m(v) = 0$, or*
2. *there are at least four vertices with multiplicity exactly 1 in L.*

Proof We show that when 2. does not hold, then 1. holds. This shows that in any case at least 1. or 2. is true.

If Case 2. does not hold, there are at most three labels with multiplicity exactly 1 in L. We now give an upper bound on the size of the set of distinct labels, that is $L^* := \{v \in V \mid \deg(v) \geq 3,\ m(v) > 0\}$. In that case, L^* is largest when there are three labels with multiplicity 1 and all other labels

occur with multiplicity 2. Note that there are exactly $2k$ edges in E_I so the total number of labels is at most $2k$. Thus, the number of labels occurring with multiplicity 2 can therefore be at most $(2k-3)/2 = k - 3/2$. Thus, $|L^*|$ is at most $k - 3/2 + 3 = k + 3/2$, and since this number needs to be an integer it is at most $k+1$, that is, $|L^*| \leq k+1$. Since there are k intervals, there are at least $n - k - |L^*|$ vertices that are neither an interval nor in L^* – these vertices are exactly those not in L. As we assume $k = n/2 - 1$, this implies that $n - k - |L^*| \geq n - k - (k+1) = 1$ and thus, there must be at least one vertex not occurring in L, as claimed by Case 1. □

In the following, we show that assuming $k < n/2$ implies that $\mu(G) > n$. To do so we use Claim 3.25 which implies that in this case at least one of 1. or 2. must be true. Therefore, we distinguish both cases of Claim 3.25 separately.

First, consider Case 1.

Lemma 3.26 *Assume $k < n/2$. If Case 1. of Claim 3.25 applies, we have $\mu(G) \geq n+1$, which is a contradiction to $\mu(G) \leq n$.*

Proof In that case, there is a vertex v such that $v \notin I$ and $v \notin L$. Hence, $\deg(v) \geq 3$. Moreover, $v \notin L$ implies that v is not adjacent to any interval. Therefore, consider the cut given by the set of intervals I. We already argued above that the size of this cut is $2k$. Now consider the cut $S = I \cup \{v\}$ obtained by adding v to I. In addition to the cut edges in I, all incident edges of v are cut edges in S, which implies $\mu(G) \geq 2k + 3 = n - 2 + 3 = n + 1$. □

We proceed with Case 2.

Lemma 3.27 *Assume $k < n/2$. If Case 2. of Claim 3.25 applies, we have $\mu(G) \geq n+1$, which is a contradiction to $\mu(G) \leq n$.*

Proof Our strategy is similar to Case 1, where we used the cut given by I and augmented it to get a cut of a size contradicting $\mu(G) \leq n$. We first observe that we can assume that there is no vertex $v \notin I$ such that $v \notin L$, as then Case 1. applies and the claim follows directly by Lemma 3.26.

To make the arguments easier to comprehend, we modify the graph G by adding an edge between the two vertices of degree 3. Note that by condition b) of irreducible walk graphs, G is semi-Eulerian but not Eulerian, implying that there are exactly two vertices of odd degree. From condition c) the only possible odd degree is 3, which implies that there are exactly two vertices of degree 3 in G. Thus, we can indeed add an edge between the vertices of degree 3. Note that if there is already an edge between the two vertices of degree 3, we just add a parallel edge. After this modification every vertex $v \in V(G)$ has a degree of $\deg(v) \in \{2, 4\}$. However, the max-cut could increase by that edge. Therefore, we show how to obtain a cut of size at least $n+2$, which then implies the claim of this lemma.

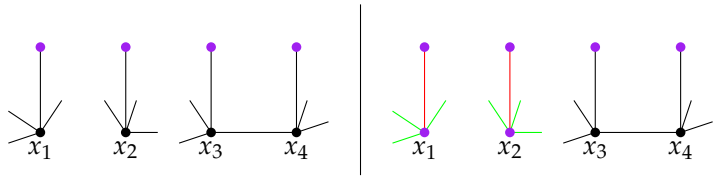


Figure 3.10: Augmenting the cut when x_1 and x_2 are non-adjacent. Both the situations before augmenting the cut given by I (left) and after the augmentation with the cut $S = I \cup \{x_1, x_2\}$ (right) are shown. The vertices in the cut are coloured purple. Edges that are lost in the cut are coloured red and edges gained in the cut are coloured green. Hence, in S we gain six additional cut edges while losing two, resulting in a total of four additional cut edges.

By Case 2. there are four vertices with multiplicity exactly 1 in L, call them x_1, x_2, x_3, x_4. We use the set $X = \{x_1, x_2, x_3, x_4\}$ to refer to these vertices. Observe that when two vertices in X are non-adjacent, say $\{x_1, x_2\} \notin E(G)$, then we can augment the cut given by I, by adding both x_1 and x_2 to this cut. This way, the cut $I \cup \{x_1, x_2\}$ will have $2k - 2 + 6 = 2k + 4 = n + 2$ cut edges, as claimed. This is also visualised in Figure 3.10.

This also shows that when there are more than four vertices with multiplicity exactly 1, we can augment the cut in the same way: Indeed, in this case, since all these vertices have degree 4, two of them must be non-adjacent. Therefore, we can assume that there are exactly four vertices with multiplicity at most 1.

Hence, we may assume that all vertices in X are pairwise adjacent. Since all $x_i \in X$ have degree $\deg(x_i) = 4$ and x_i is incident to exactly one interval, this implies that the vertices in X induce a complete subgraph. This means that we have $n > 8$, as otherwise the graph must be disconnected or containing neighbouring intervals, both are contradicting with G being an irreducible walk graph.

Moreover, we can assume that there is no vertex with multiplicity at least 3. Indeed, assuming there is such a vertex $v \notin I$ with $m(v) \geq 3$ there are $2k - 4 - 3$ edges to label with the remaining $n - k - 4 - 1$ vertices: The total number of edges to label is $2k$, four edges are labelled with vertices in X and three edges are labelled with v; also there are $n - k$ bicomponents, where four of them are used already for X and one is used for v. But then the remaining average multiplicity is $(2k - 7)/(n - k - 5) = 2 - 2/(n - 8) < 2$ for $n > 8$. Hence, there is at least one other vertex with multiplicity at most 1, which we assume not to be the case.

Hence, we can assume that there are exactly four vertices with multiplicity 1 (given by the vertices in X) and all other bicomponents have multiplicity of exactly 2. Moreover, the vertices in X induce a complete subgraph.

Let $Y = \{y_1, y_2, y_3, y_4\}$ be the intervals adjacent to x_1, x_2, x_3, x_4, respectively. First assume $|Y| \neq 4$, meaning that without loss of generality $y_1 = y_2$. Consider the cut $S = I \setminus \{y_1\} \cup \{x_1, x_2\}$. As shown in Figure 3.11 this cut

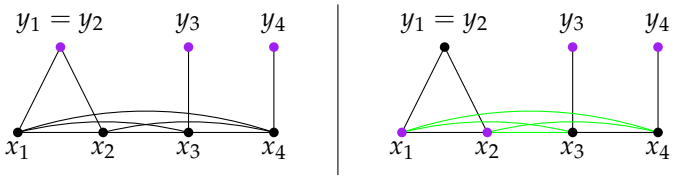


Figure 3.11: Augmenting the cut when x_1, x_2, x_3, x_4 induce a complete subgraph and $y_1 = y_2$. The left side depicts the cut I (purple vertices correspond to vertices in the cut) and the right side depicts the cut $S = I \setminus \{y_1\} \cup \{x_1, x_2\}$. There are four additional cut edges in S (coloured green).

augments the cut given by I by four edges, implying that $\mu(G) \geq 2k + 4 = n + 2$.

Next, assume $|Y| = 4$. Let $Z = \{z_1, z_2, z_3, z_4\}$ be the bicomponents adjacent to y_1, y_2, y_3, y_4, such that $z_i \neq x_i$. If $|Z| \neq 4$, we have without loss of generality $z_1 = z_2$. Consider the cut $S = I \setminus \{y_1, y_2\} \cup \{x_1, x_2, z_1\}$. As shown in Figure 3.12 this cut augments the cut given by I by six edges, implying that $\mu(G) \geq 2k + 6 \geq n + 2$.

Finally, consider the case where $|Y| = 4$ and $|Z| = 4$. Then since for all $z_i \in Z$, we have $\deg(z_i) = 4$ and $m(z_i) = 2$, there must be two non-adjacent vertices in Z, without loss of generality let them be z_1 and z_2. Then the cut given by $S = I \setminus \{y_1, y_2\} \cup \{x_1, x_2, z_1, z_2\}$ augments the cut given by I by six edges as shown in Figure 3.13. This implies that $\mu(G) \geq 2k + 6 \geq n + 2$.

Observe that this case distinction is conclusive, where in any case we have shown that $\mu(G) \geq n + 2$, exactly as claimed. □

We are now ready to prove that the number of intervals in G is given by $k = n/2$. It directly follows from Claim 3.25, Lemma 3.26 and Lemma 3.27.

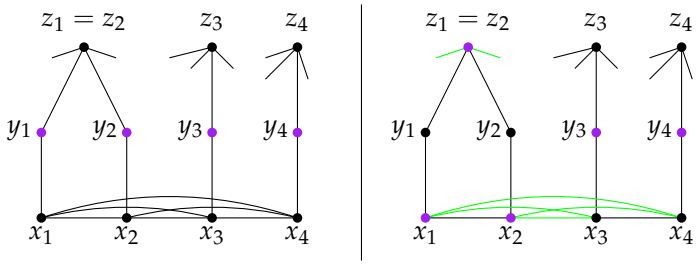


Figure 3.12: Augmenting the cut when $z_1 = z_2$. As usual, the left side depicts the cut I (with purple vertices corresponding to vertices in the cut). The right side depicts the cut $S = I \setminus \{y_1, y_2\} \cup \{x_1, x_2, z_1\}$. Note that the multiplicity of z_1 is 2, so z_1 is not adjacent to an interval other than y_1, y_2. There are six additional cut edges in S.

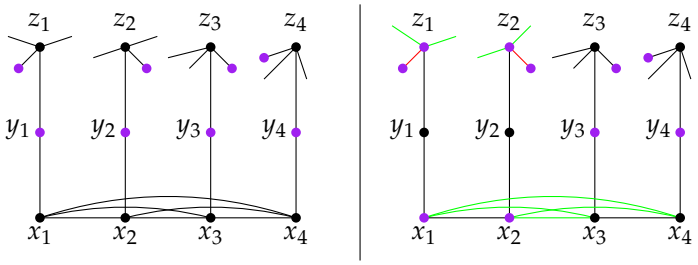


Figure 3.13: Augmenting the cut in the case when $|Y| = 4$ and $|Z| = 4$. Again, the left side depicts I with purple vertices corresponding to vertices in the cut. The right side depicts the cut $S = I \setminus \{y_1, y_2\} \cup \{x_1, x_2, z_1, z_2\}$. Observe that the vertices z_i are adjacent to some other interval different from y_i because z_i has multiplicity 2. This is depicted by the unlabelled purple vertices. We see that in S we have eight additional cut edges (green) and two edges are lost in the cut (red), resulting in a total of six additional edges.

Lemma 3.28 *Let G be an irreducible walk graph on n vertices for some even $n \geq 5$. Then the number of vertices of degree 2, denoted by k, is exactly $k = n/2$.*

Proof Lemma 3.24 implies that $k \in \{n/2 - 1, n/2\}$. Moreover, assuming $k = n/2 - 1$ we can use Claim 3.25 and apply Lemma 3.26 and Lemma 3.27 to conclude that in this case $\mu(G) \geq n + 1$, which is a contradiction to G being an irreducible walk graph. Hence, we must have $k = n/2$. □

Now that we know that the number of intervals is correct, the next step is to show that each bicomponent is adjacent to exactly two intervals in the walk graph.

Lemma 3.29 *Let G be an irreducible walk graph for some even $n \geq 5$. Then every vertex of degree at least 3 has exactly two adjacent vertices of degree 2.*

Proof First note that every bicomponent has at least one adjacent vertex of degree 2. Otherwise, use this bicomponent to augment the cut I to get a cut of size at least $2k + 3 > n$. Similarly, if there is a vertex (of degree at least 3) that has exactly one adjacent vertex of degree 2, we can use this vertex to augment the cut I as well: from Lemma 3.28 the cut size of I is exactly n and therefore adding a vertex v of degree at least 3 that is only adjacent to one interval u will increase the cut by at least one edge. The only case where this augmentation does not work is when v has two parallel edges to u. In this case, we additionally remove u from the cut.

It remains to prove that there is no bicomponent with more than two adjacent vertices of degree 2. This follows from Lemma 3.28, stating that the number of intervals, k, is given by $k = n/2$. Indeed, if there is a vertex v having three edges to intervals, there must be another vertex having exactly one adjacent interval; by the arguments above this implies $\mu(G) > n$. This can be seen by using the notation of labels again. In that case the number of distinct labels, that is $|L^*|$, can only be $n - k = k = n/2$ if there is a label with

multiplicity 1, by an analogous argument as in the proof of Claim 3.25: v has multiplicity at least 3 and hence, the total remaining multiplicity is $2k-3$ which is distributed on $k-1$ remaining bicomponents. This implies that there are at most $(2k-3)/2 = k-3/2$ vertices with multiplicity 2, showing that $|L^*| \leq k-3/2+1 < k$, implying that there must be a bicomponent with multiplicity at most 1. □

Since N_n roughly consists of a cycle and a path connecting every other vertex in that cycle, the adjacent vertices of an interval should be connected. First notice that an interval has exactly two adjacent vertices. This statement seems trivial – an interval has degree 2 after all – however, the walk graph is allowed to have parallel edges. But for irreducible necklaces, if there are two parallel edges between an interval u and some other vertex v, Lemma 3.29 implies that v has another adjacent interval u'. But then, v has multiplicity at least 3, which leads to a contradiction as seen in the proof of Lemma 3.29. This proves the following lemma.

Lemma 3.30 *Let G be an irreducible walk graph. Any vertex v with $\deg(v) = 2$ has exactly two adjacent vertices (which have degree at least 3).*

Therefore, to show that the adjacent vertices of an interval are adjacent, we prove the following lemma.

Lemma 3.31 *Let G be an irreducible walk graph for some even $n \geq 5$. For any vertex v with $\deg(v) = 2$, let v_1 and v_2 be its two adjacent vertices. If $\deg(v_1) = 4$ or $\deg(v_2) = 4$, then v_1 and v_2 are adjacent in G.*

Proof We proceed by contradiction. Assume there is an interval v with two non-adjacent neighbours v_1 and v_2, such that either v_1 or v_2 has degree 4. Consider the cut given by the set of intervals I. Its size is $2k = n$ by Lemma 3.28. Now modify this cut by removing v but inserting v_1 and v_2, that is, the cut $S = I \setminus \{v\} \cup \{v_1, v_2\}$. In this new cut, the edges (v, v_1) and (v, v_2) remain in the cut. From Lemma 3.29, we know that both v_1 and v_2 are adjacent to exactly one other interval (different from v), call them $u_1 \neq v$ and $u_2 \neq v$ respectively. Hence, in S the edges (u_1, v_1) and (u_2, v_2) are removed. However, since v_1 and v_2 are not adjacent, all other edges adjacent to them are now cut edges. Since at least one of them has degree 4, we gain at least three cut edges. Hence, S has a larger size than the cut given by I. This is a contradiction to $\mu(G) \leq n$. For an illustration of this augmentation see Figure 3.14. □

Now we are ready to prove the proposition for even n.

Proof (of Proposition 3.12, for even *n*) We build a sequence of distinct vertices $u_1, ..., u_n$ that construct a cycle in G such that for any odd $i < n-1$, there is an edge between u_i and u_{i+2}. This is exactly the graph N_n and since

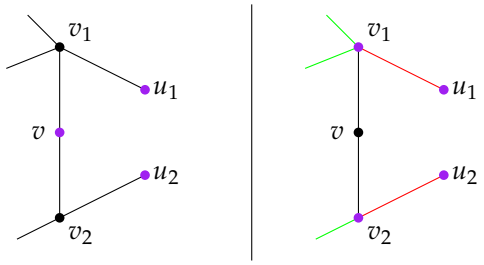


Figure 3.14: The augmentation from the proof of Lemma 3.31. The left side shows the cut I (purple vertices correspond to vertices in the cut) and the right side shows the cut given by $I \setminus \{v\} \cup \{v_1, v_2\}$. Since v_1 and v_2 are non-adjacent all edges incident to v_1 and v_2 are gained as cut edges, except the two edges incident to the intervals u_1 and u_2. As at least one of v_1, v_2 is of degree 4 – in this case v_1, we have a total of at least one additional cut edge.

both G and N_n have the same number of vertices and edges, this then implies that G is isomorphic to N_n.

To start let u_1 be an arbitrary vertex of degree $\deg(u_1) = 3$. By Lemma 3.29, the vertex u_1 is adjacent to exactly two intervals where one of which is adjacent to some vertex of degree 4. Indeed, not both of these intervals can be adjacent to only vertices of degree 3, as otherwise, we can augment the cut I by deleting these intervals and adding the vertices of degree 3 to gain two cut edges. Let u_2 be that interval and u_3 its adjacent vertex of degree 4. Note that u_1 and u_3 are adjacent by Lemma 3.31. We can now iteratively construct the sequence for every $i > 3$. To do so, we maintain the following invariant, which holds for $i \leq 3$.

(1) For all $j \leq i$ the vertices u_j are distinct,

(2) for all $j < i$ the vertices u_j and u_{j+1} are adjacent in G,

(3) for all even $j \leq i$ we have $\deg(v_j) = 2$, and

(4) for all odd $j \leq i - 2$ we have $\deg(v_j) \geq 3$, with $\deg(v_1) = 3$, and v_j is adjacent to v_{j+2}.

To construct the sequence, we assume that for any $3 \leq i < n$ we have the subsequence $u_1, \ldots, u_i$ such that the invariant holds for i and we augment this sequence to $u_1, \ldots, u_i, u_{i+1}$ that still maintains the invariant. Therefore, let $3 \leq i < n$ be arbitrary. Consider the following cases.

Case 1: $i < n$ is even Then $\deg(u_i) = 2$ (by (3)) and one of its adjacent vertices is u_{i-1} (by (2)). Let u_{i+1} be the other adjacent vertex to u_i – since u_i cannot be adjacent to any interval we have $\deg(u_{i+1}) \geq 3$. To show that the invariant still holds, we only need to show that u_{i-1} and u_{i+1} are adjacent and that $u_{i+1} \neq u_j$ for any $j < i$ – the other conditions from the invariant immediately follow. That u_{i-1} and u_{i+1} are adjacent follows from Lemma 3.31

since at least one of u_{i-1} and u_{i+1} has degree 4 (using (4) which implies that one of the two vertices of degree 3 is v_1).

To show that $u_{i+1} \neq u_j$ for any $j < i$ note that in any case, j cannot be even as otherwise, this directly contradicts the condition that no intervals are adjacent. Moreover, for any odd $1 < j < i-1$, u_j has degree 4 as it is adjacent to $u_{j-2}, u_{j-1}, u_{j+1}, u_{j+2}$ – this is implied by the invariant and Lemma 3.31 together with the observation that $j+2 < i$. So since u_i is distinct from these (using (1) and $j \leq i-3$) and u_i and u_{i+1} must be adjacent, $u_{i+1} \neq u_j$ as otherwise $\deg(u_j) > 4$. Hence, the only remaining possibility is $j = 1$ or $j = i-1$.

The latter is not possible as otherwise, u_i has only one adjacent bicomponent, contradicting Lemma 3.30. The case $j = 1$ is not possible as in this case, all edges adjacent to the vertices $u_1, \ldots, u_{i+1}$ are only adjacent to vertices in $u_1, \ldots, u_{i+1}$. However, since n is even and $i < n$ is even there is at least one vertex not occurring in $u_1, \ldots, u_{i+1}$ implying that G is not connected, which is a contradiction that G must be connected.

Case 2: $i < n$ is odd Then u_i is adjacent to the interval u_{i-1} (by (2) and (3)). Moreover, u_i is adjacent to some other interval by Lemma 3.29, let u_{i+1} be that vertex. The invariant follows, as $u_{i+1} \neq u_j$ for every $j < i$. Otherwise, since u_i is adjacent to $u_{i+1} = u_j$ either $u_i \in \{u_{j-1}, u_{j+1}\}$ – contradicting (1) of the invariant at i or implying the existence of two parallel edges between u_i and u_{i+1} – or $\deg(u_{i+1}) > 2$. The other conditions of the invariant immediately follow.

In conclusion: It remains to prove that the invariant indeed implies that the sequence $u_1, \ldots, u_n$ forms a cycle of length n and that for any odd $i < n-1$, u_i and u_{i+2} are adjacent. The latter is directly implied by (4) of the invariant. Moreover, for $i < n$ condition (2) implies that u_i and u_{i+1} are adjacent, so we need to show that u_n and u_1 are adjacent which then implies that $u_1, \ldots, u_n$ forms a cycle. The vertex u_n has two adjacent bicomponents, one of which is u_{n-1}. Since in the sequence all the vertices u_j for odd $1 < j < n-1$ have degree 4, by (2),(3),(4) and Lemma 3.31, u_n can only be adjacent to u_1. Hence, u_n and u_1 are adjacent implying that $u_1, \ldots, u_n$ forms a cycle. Since all the vertices in the sequence are distinct, the cycle is indeed of length n.

We have proven that the sequence $u_1, \ldots, u_n$ forms a cycle of length n in G such that for odd $i < n-1$ the vertices u_i and u_{i+2} are adjacent in G. Since G neither contains more vertices nor more edges, we have shown that G is isomorphic to N_n. □

This concludes the treatise on the structure of the walk graph for irreducible necklaces for even n. We have shown that for an irreducible necklace with n colours for even $n \geq 3$ the walk graph is isomorphic to N_n.

3.4.2 Proof for odd n

To see that Proposition 3.12 also holds for odd n, we use that in that case the two vertices of degree 3 are adjacent in the walk graph.

Lemma 3.32 *For an irreducible necklace C with n colours for odd n, in the walk graph G of C the two vertices of degree 3 are adjacent.*

Before we prove this lemma, let us see how this can be used to show Proposition 3.12 for odd n. We do this by reducing the odd case to the even case. In particular, from Lemma 3.32 there is an edge between the two vertices of degree 3 in the walk graph G. Replace this edge with a subdivision of that edge introducing a new vertex; we obtain a graph G' on $n+1$ vertices. The goal is to show that G' satisfies the conditions a)–e) of an irreducible walk graph. Note that a)–d) trivially hold in G'.

Therefore, we need to show that $\mu(G') \leq n+1$. Towards a contradiction assume $\mu(G') > n+1$, and let S be a maximum cut in G'. Use this same cut in G. Let v_1 and v_2 be the two vertices of degree 3. If v_1 and v_2 are on opposite sides of the cut, the cut in G will have size at least $\mu(G')$, since exactly one of the newly added edges in G' is a cut edge. This implies that $\mu(G) > n$, which is a contradiction. For the case that v_1 and v_2 are on the same side of the cut, note that every cut edge in a cut corresponds to a cut bead in the underlying necklace. Therefore, since v_1 and v_2 correspond to the first and last component of the necklace they can only be on the same side of the cut if there are an even number of cut edges. Since n is odd and we assume $\mu(G') > n+1$ this implies that $\mu(G') \geq n+3$. As the cut in G decreases by at most two edges – both the newly inserted edges in G' could be cut edges – we conclude that $\mu(G) > n$, a contradiction.

We have shown that G' is an irreducible walk graph for $n+1$ – an even number – vertices. Thus, G' is isomorphic to N_{n+1} by Proposition 3.12. Since G' is obtained by replacing an edge between the two degree 3 vertices in G by an interval and there is only one interval in N_{n+1} adjacent to both degree 3 vertices, G must be isomorphic to N_{n+1} with the interval adjacent to both degree 3 vertices replaced by an edge. But this is exactly the graph N_n. Hence, we can conclude that Proposition 3.12 also holds in the odd case.

It remains to show the lemma.

Proof (of Lemma 3.32) We prove the lemma by contradiction, so assume that the two degree 3 vertices are non-adjacent. To show the lemma, we distinguish by the number of intervals k in G – note that $k \in \{\frac{n-3}{2}, \frac{n-1}{2}\}$, as already observed in the proof of Lemma 3.28. First, we will assume $k = \frac{n-3}{2}$ and show that the two vertices of degree 3 vertices cannot be non-adjacent without leading to a contradiction and then, we show the same for $k = \frac{n-1}{2}$.

In both cases, we use a very similar strategy as in the proof of Lemma 3.28, where the assumption on the number of intervals can be used to derive two cases to distinguish the walk graph. For each of these two cases, we show that the cut given by the set of intervals, I, can be augmented to a cut of size larger than n, contradicting $\mu(G) \leq n$. In fact, we use the same terminology for labelling the incident edges of the intervals as in the proof of Claim 3.25. Recall that for an edge $e = \{u, v\}$ incident to an interval u and a bicomponent v, we label this edge with $l(e) := v$. The multiset of labels is given by $L := \{v^{m(v)} \mid \deg(v) \geq 3\}$, where the multiplicity of a bicomponent v is given by $m(v) := |\{e \in E_I \mid l(e) = v\}|$, and E_I is the set of edges incident to intervals.

Case 1: $k = (n-3)/2$ We need the following claim.

Claim 3.33 *If $k = \frac{n-3}{2}$ we have at least one of the following:*

1. *There is at least one vertex of degree 4 with multiplicity 0 in L, or*
2. *there are at least four vertices of degree 4 with multiplicity 1 in L, or*
3. *one of the vertices of degree 3 has multiplicity 0 and the other multiplicity at most 1.*

Proof (of Claim 3.33) Consider the total multiplicity of the vertices of degree 3, denoted by T. That is, when u and v are of degree 3 we have $T = m(u) + m(v)$. If $T < 2$ this implies that 3. holds. So assume $T \geq 2$. The remaining multiplicity is then at most $2k - T \leq n - 5$. Note that there are $n - k - 2 = (n-1)/2$ vertices of degree 4. However, if both 1. and 2. do not hold, the multiplicity used by vertices of degree at least 4 is at least $3 + 2 \cdot ((n-1)/2 - 3) = n - 4 > n - 5$, since there are at most three such vertices using multiplicity 1 and all other vertices use a multiplicity of at least 2. Hence, either 1. or 2. must hold. □

Now notice that this claim implies the following. In case 1. we can augment the cut given by I with that vertex of multiplicity 0 to get a cut of size $2k + 4 > n$.

For case 2. there are four vertices of degree 4 with multiplicity 1, so the cut can be increased by at least four edges using the same arguments as in the proof of Lemma 3.27. Hence, we get a cut of size at least $2k + 4 > n$.

Finally, in case 3. we can add both vertices of degree 3 vertices to the cut and since they are non-adjacent, the cut increases by at least four edges giving a cut of size at least $2k + 4 > n$.

In any case, this contradicts that the maximum cut is at most n. Hence, if $k = (n-3)/2$ the two degree 3 vertices must be adjacent.

Case 2: $k = (n-1)/2$ Similar to the previous case, we need the following claim.

Claim 3.34 *If $k = \frac{n-1}{2}$ we have at least one of the following:*

1. *There is at least one vertex with multiplicity 0 in L, or*
2. *there are at least two vertices with multiplicity 1 in L.*

Proof (of Claim 3.34) Assume 2. does not hold. Then there is at most vertex of multiplicity 1. The remaining multiplicity is then given by $2k-1$, hence the number of vertices with multiplicity 2 is at most $\frac{2k-1}{2} = k - 1/2$. Thus the number of distinct labels is at most $|L^*| \leq k - 1/2 + 1$, hence $|L^*| \leq k$. Therefore, there are at least $n - k - |L^*|$ vertices with multiplicity 0. The claim follows observing that $n - k - |L^*| \geq n - \frac{n-1}{2} - \frac{n-1}{2} = 1$. □

In case 1. of the claim, augmenting the cut I by the vertex of multiplicity 0 increases the cut by at least three edges, implying $\mu(G) > n$. In case 2. note that none of the vertices with multiplicity 1 is of degree 4, otherwise we can augment the cut I by inserting that vertex, increasing the cut by at least two edges. This implies $\mu(G) > n$. Hence, if both of the vertices with multiplicity 1 are of degree 3 since we assume they are non-adjacent, we can augment the cut I by inserting these vertices. Doing so will increase the cut by at least two edges, implying $\mu(G) > n$.

We have shown that the two degree 3 vertices are adjacent if $k = \frac{n-1}{2}$.

Since $k \in \{\frac{n-3}{2}, \frac{n-1}{2}\}$ and in both cases the degree 3 vertices are adjacent, this shows that in G the vertices of degree 3 are adjacent. □

Chapter 4

On the Computational Complexity of α-Necklace-Splitting

In Chapter 3 we showed that α-Necklace-Splitting can be solved in polynomial time. It is thus natural to ask whether the same holds for related problems, such as finding α-Ham-Sandwich cuts, or fair necklace splitting on general necklaces. Unfortunately, there is strong evidence that this is not the case. In this chapter we explore this evidence, that is, we discuss some results regarding the computational complexity of problems related to α-Necklace-Splitting.

This chapter is structured as follows. In the first section, we introduce several complexity classes that can be used to analyse the computational complexity of total search problems. Next, we state results locating problems related to α-Necklace-Splitting in these complexity classes. In particular, we show where to locate the problems Necklace-Splitting, Ham-Sandwich and α-Ham-Sandwich. Finally, we show that α-Necklace-Splitting is NP-hard when we are not given any guarantee on the separability of the necklace.

4.1 Complexity Classes for Total Search Problems

In this section, we introduce some complexity classes for total search problems. Most of these classes can be defined by a canonical complete problem for that class. That is, a class is defined by all problems that can be reduced to this canonical problem. For this thesis, it suffices to have a high-level idea of what this canonical problem is. This is why we refrain from giving formal definitions for these classes. We refer the interested reader to the corresponding literature for the formal definitions.

Total search problems are computational problems where for every instance $\mathcal{I}$ there is a non-empty set of solutions $S(\mathcal{I})$ and the goal is to find a solution $s \in S(\mathcal{I})$ given $\mathcal{I}$. The complexity class TFNP was introduced by Megiddo

and Papadimitriou [52] and is the analogue of NP for total search problems. That is, TFNP contains all total search problems where given an instance $\mathcal{I}$ and a possible solution s, it can be verified in polynomial time whether $s \in S(\mathcal{I})$. Moreover, it is required that whenever $s \in S(\mathcal{I})$, we have that $|s|$ is polynomially bounded by $|\mathcal{I}|$.

Similar to what P is to NP, the complexity class FP is the class of search problems solvable in polynomial time, which is the polynomial time counterpart of TFNP. Since any polynomial time solvable search problem can be made total by allowing a special type of solution $\perp$, which represents that the problem has no solution, we have that $\mathsf{FP} \subseteq \mathsf{TFNP}$.

As pointed out by Megiddo and Papadimitriou [52] it is not believed that TFNP has complete problems. This is the reason why Papadimitriou [55] introduced the complexity classes PPA and PPAD. The complexity class PPA (standing for *polynomial parity argument*) captures all total problems where the existence of a solution is implied by the fact that there are always an even number of vertices of odd degree in a graph G. A problem A is in PPA if it can be reduced to the following search problem in a graph G. The graph G is allowed to have exponentially many vertices, that is, $V(G) = \{0,1\}^n$, but the neighbourhood of each vertex must be computable in polynomial time, that is, in $\mathcal{O}(\mathsf{poly}(n))$ (implying that each vertex has polynomially bounded degree). Given a circuit computing the neighbourhood of each vertex, and the guarantee that $(0,\ldots,0)$ has an odd degree, the goal is to find another vertex $v \neq (0,\ldots,0)$ with an odd degree.

The complexity class PPAD (standing for *polynomial parity argument in directed graphs*) works on the analogous principle for directed graphs. It is usually modelled such that the (directed) neighbourhood of a vertex consists only of at most one predecessor and at most one successor, and the goal is to find a vertex with only an incoming edge when given a vertex with only an outgoing edge. It is easy to see that $\mathsf{PPAD} \subseteq \mathsf{PPA}$, however, it is not known if $\mathsf{PPA} = \mathsf{PPAD}$. This class has been of particular interest for algorithmic aspects of game theory, as it turned out that among others, finding Nash equilibria, market equilibria or finding Brouwer fixpoints are PPAD-complete [17, 21, 65].

Another complexity class in TFNP is PLS (*polynomial-time local search*), introduced by Johnson, Papadimitriou and Yannakakis [45]. It can be defined similarly to the classes PPA and PPAD where we are given a graph as a circuit outputting the neighbourhood of a vertex. But this time, the graph needs to be a directed acyclic graph and the goal is to find a sink [34].

A related class to PLS is CLS (*continous local search*) introduced by Daskalakis and Papadimitriou [22] as a class capturing local optimization problems with a continuous domain. They located CLS in the intersection of PPAD and PLS. Fearnley et al. showed that $\mathsf{CLS} = \mathsf{PPAD} \cap \mathsf{PLS}$ [33]. This result is of special

interest for convex optimization, as algorithms such as gradient descent or finding Karush-Kuhn-Tucker points can be shown to be CLS-complete [22].

Finally, the complexity class UEOPL (*unique end of potential line*) was introduced by Fearnley et al. [34] as a class in the intersection $\mathsf{PPAD} \cap \mathsf{PLS} = \mathsf{CLS}$. This class can again be defined by a canonical complete problem, where we are given a circuit computing the neighbourhood of a graph. This graph must be a single line, so we can assume that we are given two circuits, one computing the successor of a vertex and another computing the predecessor of a vertex. Moreover, we are given a circuit that calculates a potential for each vertex and a vertex acting as the start of the line. The goal is to find the unique end of the line, where the potential must increase along that line. This class was introduced only recently, and thus there are only a few complete problems for UEOPL [13, 34]. However, there is no known "natural" problem that is complete for UEOPL.

All of these classes are not believed to be efficiently tractable. That is, it is conjectured that for all of the discussed classes PPA,PPAD,PLS,CLS and UEOPL there is no complete problem tractable in polynomial time. In fact, there is strong evidence that this is not the case, as there are proven cryptographic hardness results for these classes. Most notably there is a randomized polynomial time reduction from integer factoring to PPA, meaning that if PPA is tractable in polynomial time, then integer factoring can be solved in randomized polynomial time, which is against the standard cryptographic belief of integer factoring being hard [44]. Furthermore, there are cryptographic hardness results for PPAD [10, 38, 57], as well as for PLS and CLS [19, 42] and even for UEOPL [9].

Figure 4.1 shows the relation of these complexity classes.

4.1.1 A detour to promise problems

Until now, many of the problems discussed are such that the totality of the problem is guaranteed by some condition implicitly assumed on the input. For example, for α-Necklace-Splitting we assume that the necklace of the input is n-separable. Considering the canonical complete problems for the complexity classes introduced above, many of these also assume some guarantee on the input. For example, PPA implicitly assumes that the vertex $(0, \ldots, 0)$ has an odd degree. Similarly, for UEOPL we assume that the underlying graph forms a single line with increasing potential. The problem formulation only treats instances that satisfy this guarantee, and it is not clear what happens to instances not satisfying this guarantee. Following the notation introduced by Even, Selman and Yacobi [32], when the problems are formulated in such a way they are called *promise problems* with the guaranteed condition being called the *promise*.

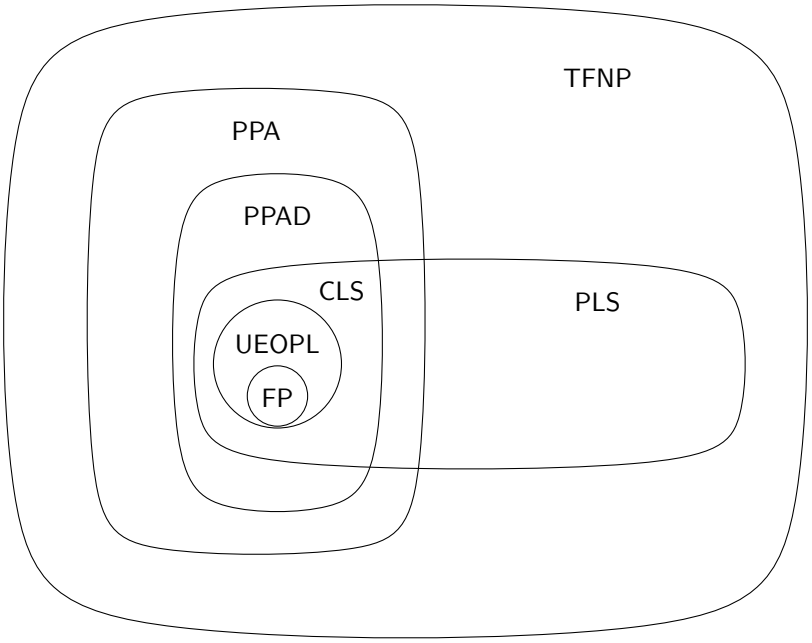


Figure 4.1: The complexity classes TFNP, PPA, PPAD, PLS, CLS,UEOPL and FP.

This leads to the natural question, of whether for a given promise problem it can be verified efficiently if an instance satisfies the promise. For example, for α-Necklace-Splitting this corresponds to the problem of deciding whether a given (arbitrary) necklace is n-separable. Borzechowski, Schnider and Weber showed that this can be done in polynomial time [15]. In fact, they even give an FPT algorithm for deciding whether a necklace is $(n-1+\ell)$-separable, using ℓ as a parameter. By Lemma 2.12 we have a direct relation between n-separable necklaces and well-separated point sets. So it is natural to ask whether the promise of α-Ham-Sandwich – the computational problem corresponding to the α-Ham-Sandwich Theorem – can also be decided in polynomial time. That is, whether deciding if a family of point sets is well-separated is possible in polynomial time. Unfortunately, this is very unlikely as Bergold et al. [8] prove coNP-completeness of this problem.

In the case where the promise cannot be verified in polynomial time, to get a problem in TFNP without relying on promises, the common strategy (used for example by Fearnley et al. [33]) is to formulate other types of solutions that correspond to violations of the promise. This strategy can also be used to make UEOPL a total search problem, which is why it can be located in TFNP. However, we want to remark that this strategy does not work for all promise problems, as it requires violations of the promise to be verifiable in polynomial time. Put in the language of complexity classes, this strategy requires a promise whose decision problem is in coNP. This also implies that

α-Ham-Sandwich is in TFNP, since deciding whether a family of point sets is well-separated is in coNP, as mentioned above.

4.2 Computational Complexity of Division Problems

In this section, we discuss some results about the computational complexity of certain division problems, related to α-Necklace-Splitting.

The foundation of the area of division problems is given by the problem Ham-Sandwich which asks to find a Ham-Sandwich cut given n discrete point sets (in general position) in $\mathbb{R}^n$. The problem Necklace-Splitting asks to find a $(\lceil |C_1|/2 \rceil, \ldots, \lceil |C_2|/2 \rceil)$-cut in a necklace $C = \{C_1, \ldots, C_n\}$ (without any separability guarantee). The totality of these problems follows from the Ham-Sandwich Theorem. Filos-Ratsikas and Goldberg [37] showed the following.

Theorem 4.1 ([37]) *Both* Necklace-Splitting *and* Ham-Sandwich *are PPA-complete.*

This result was established in a sequence of papers, starting with a result by Aisenberg, Bonet and Buss [1], showing that a search problem called 2D-Tucker is PPA-complete. The totality (and the name) of 2D-Tucker follows from Tucker's Lemma, which is a combinatorial version of the Borsuk-Ulam Theorem. For more information on Tucker's Lemma, we refer to the literature [50, 64].

Filos-Ratsikas and Goldberg then used this result to show that an approximate variant of Necklace-Splitting, called ε-Consensus-Halving, is PPA-complete [35, 36]. Finally, they reduced this approximate variant to Necklace-Splitting, implying PPA-hardness of Necklace-Splitting [37]. Since Necklace-Splitting reduces to Ham-Sandwich, by lifting the necklace to the n-dimensional moment curve (using similar arguments as we did in the proof of Proposition 2.14) this also shows that Ham-Sandwich is PPA-hard. To show PPA-completeness, they also show containment in PPA.

PPA-completeness of Necklace-Splitting and Ham-Sandiwch makes it very unlikely to find a polynomial time algorithm for these problems. Moreover, it can even be shown that decisional variants of these problems remain hard. In particular, it is NP-hard to decide whether a necklace admits a fair splitting using fewer than n cut beads [30, 53]. This also holds for ε-Consensus-Halving [23, 35]. Moreover, deciding whether there exists a Ham-Sandwich cut through a specified point is NP-hard as well [47].

The main problem in this thesis is α-Necklace-Splitting, which follows from the α-Ham-Sandwich Theorem. Furthermore, the problem α-Ham-Sandwich gets as input n well-separated discrete point sets $P_1, \ldots, P_n \subseteq \mathbb{R}^n$ and a

vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ and the task is to output an α-Ham-Sandwich cut (which always exists by the α-Ham-Sandwich Theorem). The computational complexity of α-Ham-Sandwich is still open, while the polynomial time algorithm for α-Necklace-Splitting we described in Chapter 3, together with the polynomial time algorithm that decides whether the promise holds [15], locates α-Necklace-Splitting in the complexity class FP. In fact, one motivational question of this thesis was whether hardness of α-Ham-Sandwich can be established by the hardness of α-Necklace-Splitting, similar to Ham-Sandwich and Necklace-Splitting. Our result makes it very unlikely that this question can be answered with "yes", as it is not believed that α-Ham-Sandwich can be solved in polynomial time. However, the only known result about the complexity of α-Ham-Sandwich is the following result by Chiu, Choudhary and Mulzer [18], locating it in UEOPL.

Theorem 4.2 ([18]) *α-Ham-Sandwich is in UEOPL.*

This result implies that α-Ham-Sandwich is computationally easier than Ham-Sandwich. This matches what we would suspect intuitively because α-Ham-Sandwich only works on well-separated point sets making the instances of α-Ham-Sandwich much more restricted than the Ham-Sandwich instances. Figure 4.2 puts these results into the context of the discussed complexity classes.

Similar to Necklace-Splitting, many variants of α-Necklace-Splitting are hard to solve. In particular, Epping, Hochstättler and Oertel [30] showed that for arbitrary necklaces it is NP-hard to determine the minimal number of cut points to get an α-splitting of the necklace. In their setting, a different notation is used (they work on the *paint shop problem for words*), however, it can be easily translated to necklaces where the cut points can be chosen arbitrarily, even multiple of the same colour. We will see in the next section that it is NP-hard to decide if there is an α-cut (as per Definition 2.13) in an arbitrary necklace. This result makes it unlikely to find an algorithm that efficiently finds α-cuts for necklaces without any condition on the separability.

4.3 NP-Hardness of α-Necklace-Splitting in General Necklaces

In this section, we show NP-hardness of deciding whether an arbitrary necklace has an α-cut. However, since for arbitrary necklaces α-cuts may not be unique, it is rather unnatural to define the positive side of a cut via its cut parity. Therefore, in this section, an α-cut is defined as in Definition 2.13, but the positive side can be chosen arbitrarily, not necessarily the side induced by the cut parity. We will show NP-hardness (in fact even completeness) of the following problem.

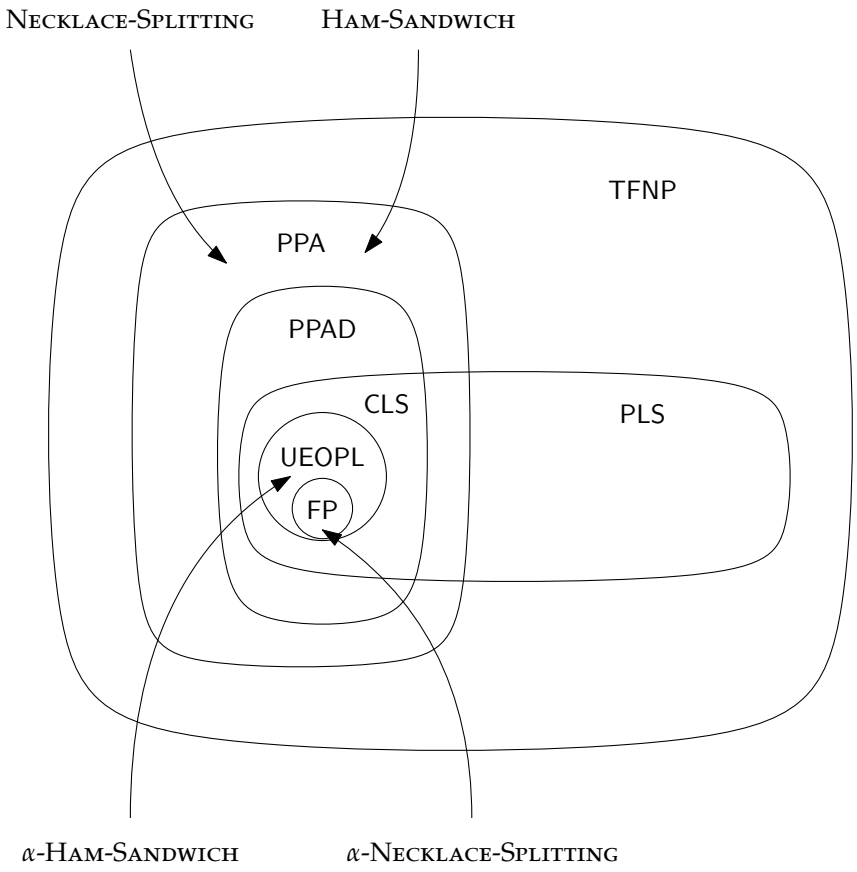


Figure 4.2: The problems Necklace-Splitting, Ham-Sandwich,α-Necklace-Splitting, α-Ham-Sandwich located in complexity classes. An arrow indicates that a problem is in this complexity class. Note that for Necklace-Splitting and Ham-Sandwich the problems are even complete for PPA.

Definition 4.3 α-Necklace-Deciding

Input: *A necklace* $C = \{C_1, \ldots, C_n\}$ *and* $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{1, \ldots, |C_i|\}$
Decide: *Does C have an* α*-cut?*

Note that hardness of this problem also implies hardness of the same problem with α-cuts defined using the positive side induced by the cut parity. This can be seen as the problem with the cut parity can just be solved twice, once for α and once for $\overline{\alpha}$ to decide if an α-cut with arbitrarily chosen positive side exists.

Proposition 4.4 α-Necklace-Deciding *is NP-complete.*

Proof A valid α-cut can be used as a certificate and it is polynomial time verifiable if a given α-cut is indeed valid. This shows membership of NP.

To show NP-hardness we reduce from 3-Sat. Let $\Phi = C_1 \wedge \cdots \wedge C_m$ be an instance of 3-Sat. Each clause C_i consists of exactly three variables. We now

construct a necklace ζ and a vector α such that ζ has an α-cut if and only if Φ is satisfiable. We construct ζ by giving a string corresponding to how the beads occur sequentially from left to right.

To do this, we use the following types of beads. Two types of beads P and N will be used to enforce if parts of the necklace need to be on the positive or negative side. For every variable x, we use the types x_0^A and x_0^B which will be used to construct a gadget which encodes if x is set to true or false. For every clause C_i such that $x \in C_i$, we will additionally use the types x_i^A and x_i^B which will be used to read out the value of x at the clause C_i. To transfer the assignment of x, we use yet another type of bead x_T.

For clauses C_i we only need one type of bead, which we simply name C_i. Moreover, we need multiple types for separator beads that only occur once, we denote them by S_* where the subscript indicates to what part of the necklace this separator bead belongs to.

Finally, we need two types of beads a, b that are only used for a technical detail enforcing where to place the positive side of the cut. The necklace starts with the string $a\,b\,a$ where we want both beads of a on the positive side. This way, the cut is forced to have the part to the left of the necklace on its positive side. Indeed, since there must be exactly one cut through one bead of a and another cut goes through b, the only way of moving both a beads to the positive side is by choosing the positive side such that the part to the left of the necklace is on the positive side.

The necklace then consists of two parts, each of which consists of multiple parts. The first part is the encoding part where we encode the assignment of variables and the satisfiability of each clause. In the second part, we need to enforce the position of cuts for some types of beads to ensure that an α-cut in the necklace indeed corresponds to a satisfiable assignment of Φ. In the following, we describe both parts, starting with the encoding part.

For the encoding part, we have again two parts, a variable and a clause part. In the variable part, we add the following string for each variable x

$$P\, x_0^A\, \underbrace{x_T \dots x_T}_{k \text{ times}}\, x_0^B\, P\, x_0^A\, x_0^B\, P$$

where k is the number of occurrences of the variable x, that is, the number of clauses C_i such that $x \in C_i$. For the clause part, we add the following string for each clause C_i and each variable $x \in C_i$.

$$\begin{array}{llllll} P\, x_i^A & C_i & x_i^B\, P\, x_i^A\, x_T & & x_i^B\, P & \text{if } x \text{ appears as a positive literal in } C_i, \\ P\, x_i^A & & x_i^B\, P\, x_i^A\, x_T & C_i & x_i^B\, P & \text{if } x \text{ appears as a negative literal in } C_i. \end{array}$$

Note that the only difference between these two strings is the placement of the bead of type C_i.

The intuition behind the encoding part is the following. Assume there is no cut in any bead of types P, x_T, C_i within the encoding part – this will be enforced in the second part of the necklace. The α-vector is such that on the positive side of the cut, we have all beads of type P, half of each x_T, exactly three beads of type C_i and both beads of types x_i^A and x_i^B (for all i including 0). Then the only way to make the x_0^A and x_0^B types having both beads on the positive side is by either cutting through the first x_0^A and first x_0^B or by cutting through the second x_0^A and second x_0^B. Otherwise, either α is not satisfied for x_0^A, x_0^B, or the bead of type P between the first occurrence of x_0^B and the second occurrence of x_0^A is on the negative side implying that α is not satisfied for P. The first case will encode the assignment $x =$ true and the latter encodes $x =$ false.

The cuts corresponding to a true assignment will move the x_T between x_0^A and x_0^B to the negative side. Since we need half of the beads x_T on the positive side, this implies that for the cuts in x_i^A and x_i^B for all clauses C_i containing x the bead of type x_T must be on the positive side. Hence, the cuts in x_i^A and x_i^B are also corresponding to the assignment of x, in the same way as the cuts in x_0^A and x_0^B do.

The beads of type C_i are thus on the negative side for a variable assigned to a satisfying assignment for that clause. That is, for a variable x such that the cuts in x_0^A and x_0^B encode an assignment of x such that C_i is satisfied by that assignment, the bead C_i between the beads x_i^A and x_i^B will be on the negative side of the cut. Hence, if there is a satisfiable assignment for Φ there is a choice of cuts – corresponding to the satisfiable assignment – such that each C_i has at least one bead on the negative side (or equivalently, at most two on the positive side). In the second part of the necklace, we will enforce that the cut in C_i will be in the second part, implying that we can move exactly three beads of type C_i to the positive side of the cut.

Now we construct the second part of the necklace, which controls the cuts. For each clause C_i, we add the following string

$$P\, C_i\, C_i\, C_i\; S_i$$

where S_i is a new separator bead. It will be enforced that the cut for C_i will be in this string. We need to add three additional beads of type C_i so that at least one but at most three beads of C_i can be moved to the positive side. Hence, we can only have exactly three beads of that type on the positive side if there are at most two beads already on the positive side from the encoding part – implying that the clause C_i is satisfiable by at least one variable appearing in it.

For the variables, we add the following string for each variable x

$$\underbrace{P\, x_T\; \ldots\; P\, x_T}_{k \text{ times}} \underbrace{N\, x_T\; \ldots\; N\, x_T}_{k \text{ times}}\; N\, S_x$$

where again k is the number of clauses containing x. This string will make sure that x_T is halved.

Finally, we need to add the following string to control the cuts in P and N.

$$P\,N$$

We give an example of this construction at the end of this section.

Now the necklace ζ is given by the concatenation of these strings in order as we described them. Note that the number of types of beads is even: P comes paired with N, x_i^A comes paired with x_i^B, C_i is paired with S_i and x_T is paired with S_x. Also, recall the definition of our α-vector. We will use the notation $\alpha(X)$ to denote the value corresponding to the type of bead X and $|X|$ to denote the number of beads of type X occurring in ζ. We have

$$\begin{aligned}
&\alpha(a) = 2,\ \alpha(b) = 1,\\
&\alpha(P) = |P|,\ \alpha(N) = 1,\\
&\alpha(x_i^A) = 2,\ \alpha(x_i^B) = 2 && \text{for all variables } x \text{ and all } i \text{ including } 0,\\
&\alpha(x_T) = \frac{|x_T|}{2} && \text{for all variables } x_T\text{, note that } |x_T| \text{ is even,}\\
&\alpha(C_i) = 3 && \text{for all clauses } C_i,\\
&\alpha(S_*) = 1 && \text{for any separator } S_*.
\end{aligned}$$

Now note that a satisfiable assignment of Φ corresponds to an α-cut in ζ as follows. For a variable x with an assignment of true, cut through the first occurrences of x_i^A and x_i^B for all i including 0. Similarly, for an assignment of false, cut through the second occurrences of x_i^A and x_i^B. The cut for x_T is placed in the middle bead x_T of the string (that is, between the last bead of type P and the first bead of type N)

$$P\,x_T\,\ldots\,P\,x_T\,N\,x_T\,\ldots\,N\,x_T\,N\,S_x.$$

In this way, the requirements of α are already satisfied for all x_i^A, x_i^B, x_T. Moreover, by cutting through each separator point we get the (trivial) requirements for the separator points.

Now note that when every clause is satisfiable, the beads C_i appearing between some x_i^A and x_i^B have at most two beads on the positive side since there must be at least one variable with a satisfiable assignment for that clause. Hence, we can place a cut at the correct spot in the string $P\,C_i\,C_i\,C_i\,S_i$ to get three beads of C_i on the positive side. Also note that except the one from the very last part of the necklace, every bead of type P is on the positive side of the cut and every bead of type N is on the negative side of the cut by construction of ζ with the placement of the separator beads. Since there are an even number of bead types, we can cut through the last occurrence of P

and the last occurrence of N to also satisfy the requirements on these types. Hence, if there is a satisfiable assignment of Φ there is an α-cut of ζ.

For the other direction, we need to show that when ζ has an α-cut, there is a satisfiable assignment of Φ. Note that the starting string $a\,b\,a$ ensures that the part to the left (and to the right) of ζ must be positive. Moreover, observe that the cuts in P and N must be in the very last part of ζ, that is in the string $P,\ N$. Indeed, if, say, P is cut earlier, since the last bead P must be on the positive side and there are an even number of cuts (as there are an even number of types of beads) N must also be cut before the last part. But then the last N is on the positive side, which is not an α-cut. Similarly, if N is cut before the last part, the last N is on the positive side since there are an even number of cuts.

But this implies that the cut for x_T must be placed in the middle bead of the string $P\,x_T\,\ldots\,P\,x_T\,N\,x_T\,\ldots\,N\,x_T\,N\,S_x$, as otherwise the beads of type P and N are on the wrong sides of the cut. Moreover, the cut for C_i must be in the string $P\,C_i\,C_i\,C_i\,S_i$. Otherwise, since P must be on the positive side, all three C_i are on the positive side, which implies that there are four C_i on the positive side (as the cut bead is also counted). This is not an α-cut.

These observations imply that between any beads x_i^A and x_i^B, there is no cut (except the ones for x_i^A and x_i^B), as between the beads x_i^A and x_i^B there are only beads of types P, C_i and x_T. Hence, there are only two ways of cutting the beads x_i^A and x_i^B, namely either through the first occurrences of both types or through the second occurrences. The first choice will correspond to a true assignment of x and the latter to a false assignment. Since the beads x_T must be halved, this makes sure that the choice of these cuts is consistent among all i.

To see that this cut corresponds to a satisfiable assignment of Φ, consider a clause C_i. Since there are three beads of C_i on the positive side, there must be at least one variable $x \in C_i$ such that the corresponding bead C_i between x_i^A and x_i^B is on the negative side. By construction of ζ, the cuts for x_i^A and x_i^B are thus corresponding to a satisfiable assignment of x for C_i. Since this holds for all clauses this shows that Φ is satisfied.

Now observe that the necklace ζ is of size polynomial in the number of clauses of Φ and thus, α-Necklace-Deciding is NP-hard. □

We conclude this section by giving an example of the construction used in this proof. Consider the 3-Sat instance $\Phi = C_1 \wedge C_2$ where $C_1 = x \vee \overline{y} \vee z$ and $C_2 = x \vee \overline{z} \vee w$. Here, we use $\overline{y}$ to denote a negative literal of variable y.

There are four variables, and the variable parts are given by:

$$P\, x_0^A\, x_T\, x_T\, x_0^B\, P\, x_0^A\, x_0^B\, P$$
$$P\, y_0^A\, y_T\, y_0^B\, P\, y_0^A\, y_0^B\, P$$
$$P\, z_0^A\, z_T\, z_T\, z_0^B\, P\, z_0^A\, z_0^B\, P$$
$$P\, w_0^A\, w_T\, w_0^B\, P\, w_0^A\, w_0^B\, P$$

The clause parts are given by, first for clause C_1

$$P\, x_1^A\, C_1\, x_1^B\, P\, x_1^A\, x_T\, x_1^B\, P$$
$$P\, y_1^A\, y_1^B\, P\, y_1^A\, y_T\, C_1\, y_1^B\, P$$
$$P\, z_1^A\, C_1\, z_1^B\, P\, z_1^A\, z_T\, z_1^B\, P$$

and for C_2

$$P\, x_2^A\, C_2\, x_2^B\, P\, x_2^A\, x_T\, x_2^B\, P$$
$$P\, z_2^A\, z_2^B\, P\, z_2^A\, z_T\, C_2\, z_2^B\, P$$
$$P\, w_2^A\, C_2\, w_2^B\, P\, w_2^A\, w_T\, w_2^B\, P$$

The enforcing part starts with the enforcement of the clauses by

$$P\, C_1\, C_1\, C_1\, S_1$$
$$P\, C_2\, C_2\, C_2\, S_2$$

and for the variables

$$P\, x_T\, P\, x_T\, N\, x_T\, N\, x_T\, N\, S_x$$
$$P\, y_T\, N\, y_T\, N\, S_y$$
$$P\, z_T\, P\, z_T\, N\, z_T\, N\, z_T\, N\, S_z$$
$$P\, w_T\, N\, w_T\, N\, S_w$$

Finally, the enforcement of P and N is given by

$$P\, N$$

In conclusion, the necklace is given by the following string (line breaks and spacing are inserted for better readability, and to emphasize the structure of the construction). Beads in bold face show the cuts corresponding to the

(satisfying) assignment $x =$ "false", $y =$ "false", $z =$ "true", $w =$ "true".

$\boldsymbol{a}\,\boldsymbol{b}\,a$

$P\,x_0^A\,x_T\,x_T\,x_0^B\,P\,\boldsymbol{x}_0^A\,\boldsymbol{x}_0^B\,P \quad P\,y_0^A\,y_T\,y_0^B\,P\,\boldsymbol{y}_0^A\,\boldsymbol{y}_0^B\,P \quad P\,\boldsymbol{z}_0^A\,z_T\,z_T\,\boldsymbol{z}_0^B\,P\,z_0^A\,z_0^B\,P$
$P\,\boldsymbol{w}_0^A\,w_T\,\boldsymbol{w}_0^B\,P\,w_0^A\,w_0^B\,P$

$P\,x_1^A\,C_1\,x_1^B\,P\,\boldsymbol{x}_1^A\,x_T\,\boldsymbol{x}_1^B\,P \quad P\,y_1^A\,y_1^B\,P\,\boldsymbol{y}_1^A\,y_T\,C_1\,\boldsymbol{y}_1^B\,P \quad P\,\boldsymbol{z}_1^A\,C_1\,\boldsymbol{z}_1^B\,P\,z_1^A\,z_T\,z_1^B\,P$
$P\,x_2^A\,C_2\,x_2^B\,P\,\boldsymbol{x}_2^A\,x_T\,\boldsymbol{x}_2^B\,P \quad P\,\boldsymbol{z}_2^A\,\boldsymbol{z}_2^B\,P\,z_2^A\,z_T\,C_2\,z_2^B\,P \quad P\,\boldsymbol{w}_2^A\,C_2\,\boldsymbol{w}_2^B\,P\,w_2^A\,w_T\,w_2^B\,P$

$P\,C_1\,\boldsymbol{C}_1\,C_1\,\boldsymbol{S}_1 \quad P\,\boldsymbol{C}_2\,C_2\,C_2\,\boldsymbol{S}_2$

$P\,x_T\,P\,\boldsymbol{x}_T\,N\,x_T\,N\,x_T\,N\,\boldsymbol{S}_x \quad P\,\boldsymbol{y}_T\,N\,y_T\,N\,\boldsymbol{S}_y \quad P\,z_T\,P\,\boldsymbol{z}_T\,N\,z_T\,N\,z_T\,N\,\boldsymbol{S}_z$
$P\,\boldsymbol{w}_T\,N\,w_T\,N\,\boldsymbol{S}_w$

$\boldsymbol{P}\,\boldsymbol{N}$

Moreover, α is given as

$\alpha(a) = 2, \alpha(b) = 1$
$\alpha(P) = 39 = |P|, \alpha(N) = 1$
$\alpha(x_0^A) = \alpha(x_0^B) = \alpha(x_1^A) = \alpha(x_1^B) = \alpha(x_2^A) = \alpha(x_2^B) = 2$
$\alpha(y_0^A) = \alpha(y_0^B) = \alpha(y_1^A) = \alpha(y_1^B) = 2$
$\alpha(z_0^A) = \alpha(z_0^B) = \alpha(z_1^A) = \alpha(z_1^B) = \alpha(z_2^A) = \alpha(z_2^B) = 2$
$\alpha(w_0^A) = \alpha(w_0^B) = \alpha(w_2^A) = \alpha(w_2^B) = 2$
$\alpha(x_T) = 4 = \frac{|x_T|}{2}, \alpha(y_T) = 2 = \frac{|y_T|}{2}, \alpha(z_T) = 4 = \frac{|z_T|}{2}, \alpha(w_T) = 2 = \frac{|w_T|}{2}$
$\alpha(C_1) = \alpha(C_2) = 3$
$\alpha(S_1) = \alpha(S_2) = \alpha(S_x) = \alpha(S_y) = \alpha(S_z) = \alpha(S_w) = 1.$

Chapter 5

Outlook

This final chapter is dedicated to pointing to further directions for future research. In particular, we want to highlight some areas we explored during the work of this thesis.

5.1 FPT Algorithm on $(n-1+\ell)$-Separable Necklaces

While the NP-hardness of α-Necklace-Deciding makes it unlikely to find efficient algorithms for finding α-cuts (if they exist) in general necklaces, we may still find efficient algorithms if we have some condition on the separability. In particular, similarly to the FPT algorithm for Necklace-Splitting from Borzechowski, Schnider and Weber [15] working on $(n-1+\ell)$-separable necklaces, we may find an FPT algorithm for α-Necklace-Splitting on $(n-1+\ell)$-separable necklaces, using ℓ as a parameter.

In fact, we suppose that a slight modification of the algorithm we described in Chapter 3 can be used. In the following, we describe our ideas on a high level, without dealing with the details. The only problem with directly using that algorithm is that some recursive steps may find an α-cut in the smaller instance that cannot be augmented to an α-cut in the original necklace. Namely, the step from the first phase when iterating over all possible components of colours with more than two components and identifying an α-cut in a necklace when only considering the components of the current iteration may fail: Since α-cuts may not be unique, the recursive step could find an α-cut in the smaller necklace that cannot be used to augment to an α-cut in the original necklace. Similarly, the step in the second phase where the first and last components are of the same colour may fail because of the same issue of the recursive call finding another α-cut than the one that can be used to augment to an α-cut in the original necklace.

However, the other steps of the first and second phases still work. In particu-

lar, removing neighbouring intervals is still possible, as we know where to put the cuts in there. The modified algorithm will then perform only these steps that still work and just skip the steps that may fail. We can still apply the ideas from the third phase, by solving an integer linear program (which needs to be slightly adapted from the formulation we give in Chapter 3, but this is straightforward). Since we did not remove the colours of many components, the primal graph of this ILP can contain cliques of size as large as the number of components of any colour. By adapting Lemma 3.4 and Lemma 3.3 we can show that there are at most $\mathcal{O}(\ell)$ colours with more than two components and each of these has at most $\mathcal{O}(\ell)$ components. Therefore, it is straightforward to show that the treewidth of the primal graph of this ILP is bounded by $\mathcal{O}(\ell \cdot \mathsf{tw}(G))$ where G is the colour graph obtained by applying the modified algorithm.

In Chapter 3 we argued that when performing the regular algorithm, G is the graph of an irreducible necklace. However, since we do not apply all the steps from the algorithm, the necklace may not be irreducible. As we are only interested in the treewidth of the graph G, we claim that we can modify G to obtain a graph G' which is the graph of an irreducible necklace and $\mathsf{tw}(G) \in \mathcal{O}(\mathsf{poly}(\ell) + \mathsf{tw}(G'))$. To see this, we need to define irreducible $(n-1+\ell)$-separable necklaces. This can be done analogously as for n-separable necklaces, where we only need to change the separability condition to be $\mathsf{sep}(C) \leq n-1+\ell$. The walk graph of these necklaces can then be characterized as follows.

Lemma 5.1 *Let C be an irreducible $(n-1+\ell)$-necklace. Its walk graph $G = (V, E)$ satisfies all of the following conditions.*

a) G is connected,

b) G is semi-Eulerian but not Eulerian,

c) for all vertices $v \in V$ we have $\deg(v) \in \{2, 3, 4\}$,

d) there are no adjacent vertices of degree 2,

e) the maximum cut of G is at most $\mu(G) \leq n-1+\ell$.

Now to see how to transform G to a walk graph of an irreducible necklace, note that when performing the modified algorithm, G will already satisfy conditions a) (trivial), d) (removing neighbouring intervals) and e) (the necklace stays $(n-1+\ell)$-separable). To satisfy condition c) we need to remove all components of colours with more than two components. Moreover, we need to remove neighbouring intervals that are possibly introduced by removing these components. We already stated above that there are only $\mathcal{O}(\mathsf{poly}(\ell))$ components (and thus additional intervals) to remove. It is easy to see that each removal decreases the treewidth only by a constant.

Similarly, to satisfy condition b) the first and last components need to be of different colours. We can simply remove the whole colour in the graph, however, it may still have the first and last components of the same colour. It can be shown that there can only be $\mathcal{O}(\ell)$ such nested first and last components of the same colour. Moreover, this removal only decreases the treewidth by a constant.

Therefore, when applying these steps to G, we obtain a walk graph G' for an irreducible necklace. If $\mathsf{tw}_\ell(G_{IRR})$ is the maximum treewidth among all irreducible $(n-1+\ell)$-separable necklaces, we thus have $\mathsf{tw}(G) \in \mathcal{O}(\mathsf{poly}(\ell) + \mathsf{tw}_\ell(G_{IRR}))$. This is where the analysis fails since we do not know how large $\mathsf{tw}_\ell(G_{IRR})$ is. We conjecture that for irreducible $(n-1+\ell)$-separable necklaces, their walk graphs have treewidth only depending on ℓ, which would imply that our algorithm is indeed an FPT algorithm using ℓ as a parameter.

Conjecture 5.2 *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for any irreducible $(n-1+\ell)$-separable necklace its walk graph G has $\mathsf{tw}(G) \leq f(\ell)$. In particular, we have $\mathsf{tw}_\ell(G_{IRR}) \leq f(\ell)$.*

Examining this conjecture and delving into the details of our modified algorithm may be part of future research.

5.2 Generalizing for Higher Dimensions

A necklace is a family of one-dimensional points. Moreover, an α-cut consists of n points, which can be viewed as hyperplanes in $\mathbb{R}$. Therefore, it is a reasonable question whether we can generalize α-Necklace-Splitting to higher dimensions. That is, a necklace consists of n families of point sets in $\mathbb{R}^d$ and an α-cut consists of an arrangement of hyperplanes in $\mathbb{R}^d$ partitioning the point sets according to α. As a first step, the notion of separability can be translated to higher dimensions. For a family of point sets $\mathcal{P} = \{P_1, \ldots, P_n\}$ with $P_i \subseteq \mathbb{R}^d$ the separability is given by the smallest integer k such that any choice of point sets $\mathcal{S} \subseteq \mathcal{P}$ can be separated from $\mathcal{P} \setminus \mathcal{S}$ by an arrangement of k hyperplanes.

We empirically verified the following conjecture (by enumeration using a computer program), but have not found a rigorous proof.

Conjecture 5.3 *Any seven points in $\mathbb{R}^2$ cannot be 2-separable.*

Moreover, during the work of this thesis, we tried to find a variant of Proposition 2.14 in two dimensions. That is, we tried to find sufficient conditions of $n = 4$ point sets in $\mathbb{R}^2$ such that for all α an α-cut exists, where an α-cut is given by an arrangement of two lines. However, we did not manage to find such conditions.

5.3 Computational Complexity of α-Ham-Sandwich

In the realm of computational complexity of problems related to α-Necklace-Splitting, the major open problem is to resolve the complexity of α-Ham-Sandwich. Our algorithm from Chapter 3 makes it unlikely that hardness of α-Ham-Sandwich can be shown by proving hardness of α-Necklace-Splitting. We only looked briefly into the area of the computational complexity of α-Ham-Sandwich and related problems. There is still ample space to explore in future work.

Bibliography

[1] James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-D Tucker is PPA complete. *Journal of Computer and System Sciences*, 108:92–103, 2020. doi:10.1016/j.jcss.2019.09.002.

[2] Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987. doi:10.1016/0001-8708(87)90055-7.

[3] Noga Alon and Andrei Graur. Efficient Splitting of Necklaces. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.14.

[4] Noga Alon and Douglas B. West. The Borsuk-Ulam theorem and bisection of necklaces. In *Proceedings of the American Mathematical Society*, volume 98, pages 623–628. American Mathematical Society, 1986. doi:10.1090/S0002-9939-1986-0861764-9.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009. URL: https://theory.cs.princeton.edu/complexity/.

[6] Imre Bárány, Alfredo Hubard, and Jesús Jerónimo. Slicing Convex Sets and Measures by a Hyperplane. *Discrete & Computational Geometry*, 39(1):67–75, 2008. doi:10.1007/s00454-007-9021-2.

[7] Luis Barba, Alexander Pilz, and Patrick Schnider. Sharing a pizza: bisecting masses with two cuts, 2019. doi:10.48550/arXiv.1904.02502.

[8] Helena Bergold, Daniel Bertschinger, Nicolas Grelier, Wolfgang Mulzer, and Patrick Schnider. Well-Separation and Hyperplane Transversals

in High Dimensions. In Arthus Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2022.16.

[9] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D. Rothblum. PPAD is as Hard as LWE and Iterated Squaring. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, pages 593–622, Cham, 2022. Springer Nature Switzerland. doi:10.1007/978-3-031-22365-5_21.

[10] Nir Bitansky, Omer Paneth, and Alon Rosen. On the Cryptographic Hardness of Finding a Nash Equilibrium. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1480–1498, 2015. doi:10.1109/FOCS.2015.94.

[11] Paul Bonsma, Thomas Epping, and Winfried Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Applied Mathematics*, 154(9):1335–1343, 2006. 2nd Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2003). doi:10.1016/j.dam.2005.05.033.

[12] Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933. URL: http://eudml.org/doc/212624.

[13] Michaela Borzechowski. *The Complexity Class Unique End of Potential Line*. Master's thesis, Freie Universität Berlin, 2021. URL: https://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Borzechowski21.pdf.

[14] Michaela Borzechowski, John Fearnley, Spencer Gordon, Rahul Savani, Patrick Schnider, and Simon Weber. Two Choices Are Enough for P-LCPs, USOs, and Colorful Tangents. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2024.32.

[15] Michaela Borzechowski, Patrick Schnider, and Simon Weber. An FPT Algorithm for Splitting a Necklace Among Two Thieves. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on*

Algorithms and Computation (ISAAC 2023), volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2023.15.

[16] Steven J. Brams and Alan D. Taylor. An Envy-Free Cake Division Protocol. *The American Mathematical Monthly*, 102(1):9–18, 1995. doi:10.1080/00029890.1995.11990526.

[17] Xi Chen and Xiaotie Deng. Settling the Complexity of Two-Player Nash Equilibrium. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 261–272, 2006. doi:10.1109/FOCS.2006.69.

[18] Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational Complexity of the α-Ham-Sandwich Problem. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.31.

[19] Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a Nash equilibrium is no easier than breaking Fiat-Shamir. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 1103–1114, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316400.

[20] Richard Cole, Micha Sharir, and Chee K. Yap. On k-hulls and related problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 154–166, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800057.808677.

[21] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

[22] Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, page 790–804, USA, 2011. Society for Industrial and Applied Mathematics. doi:10.5555/2133036.2133098.

[23] Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving

and the Borsuk-Ulam theorem. *Journal of Computer and System Sciences*, 117:75–98, 2021. doi:10.1016/j.jcss.2020.10.006.

[24] Xiaotie Deng, Qi Qi, and Amin Saberi. Algorithmic solutions for envy-free cake cutting. *Operations Research*, 60(6):1461–1476, 2012. doi:10.1287/opre.1120.1116.

[25] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, Heidelberg, 1987. doi:10.1007/978-3-642-61568-9.

[26] Herbert Edelsbrunner and Roman Waupotitsch. Computing a ham-sandwich cut in two dimensions. *Journal of Symbolic Computation*, 2(2):171–178, 1986. doi:10.1016/S0747-7171(86)80020-7.

[27] Herbert Edelsbrunner and Emo Welzl. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Information Processing Letters*, 23(5):289–293, 1986. doi:10.1016/0020-0190(86)90088-8.

[28] Christopher S. Edwards. Some Extremal Properties of Bipartite Subgraphs. *Canadian Journal of Mathematics*, 25(3):475–485, 1973. doi:10.4153/CJM-1973-048-x.

[29] Christopher S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Proceedings of the 2nd Czechoslovak Symposium on Graph Theory*, pages 167–181, Prague, 1975.

[30] Thomas Epping, Winfried Hochstättler, and Peter Oertel. Complexity results on a paint shop problem. *Discrete Applied Mathematics*, 136(2):217–226, 2004. The 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization. doi:10.1016/S0166-218X(03)00442-6.

[31] Paul Erdős. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3:113–116, 1965. doi:10.1007/BF02760037.

[32] Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984. doi:10.1016/S0019-9958(84)80056-X.

[33] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 46–59, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451052.

[34] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020. doi:10.1016/j.jcss.2020.05.007.

[35] Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness Results for Consensus-Halving. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2018.24.

[36] Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 51–64, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188880.

[37] Aris Filos-Ratsikas and Paul W. Goldberg. The Complexity of Necklace Splitting, Consensus-Halving, and Discrete Ham Sandwich. *SIAM Journal on Computing*, 52(2):200–268, 2023. doi:10.1137/20M1312678.

[38] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 579–604, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. doi:10.1007/978-3-662-53008-5_20.

[39] Charles H. Goldberg and Douglas B. West. Bisection of Circle Colorings. *SIAM Journal on Algebraic Discrete Methods*, 6(1):93–106, 1985. doi:10.1137/0606010.

[40] Charles R. Hobby and John R. Rice. A Moment Problem in L1 Approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965. doi:10.2307/2033900.

[41] Alfredo Hubard and Roman Karasev. Bisecting measures with hyperplane arrangements. *Mathematical Proceedings of the Cambridge Philosophical Society*, 169(3):639–647, 2020. doi:10.1017/S0305004119000380.

[42] Pavel Hubáček and Eylon Yogev. Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds. *SIAM Journal on Computing*, 49(6):1128–1172, 2020. doi:10.1137/17M1118014.

[43] Bart M. P. Jansen and Stefan Kratsch. A Structural Approach to Kernels for ILPs: Treewidth and Total Unimodularity. In Nikhil Bansal

and Irene Finocchi, editors, *Algorithms - ESA 2015*, pages 779–791, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-48350-3_65.

[44] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016. doi:10.1016/j.jcss.2015.08.001.

[45] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

[46] Duško Jojić, Gaiane Panina, and Rade Živaljević. Splitting Necklaces, with Constraints. *SIAM Journal on Discrete Mathematics*, 35(2):1268–1286, 2021. doi:10.1137/20M1331949.

[47] Christian Knauer, Hans Raj Tiwary, and Daniel Werner. On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems. In *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, volume 9, pages 649–660, Dortmund, Germany, 2011. URL: https://hal.science/hal-00573613.

[48] Chi-Yuan Lo, Jiří Matoušek, and William Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11(4):433–452, 1994. doi:10.1007/BF02574017.

[49] Jiří Matoušek. *Lectures on Discrete Geometry*. Springer, New York, NY, 2002. doi:10.1007/978-1-4613-0039-7.

[50] Jiří Matoušek. *Using the Borsuk-Ulam Theorem*. Springer, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-76649-0.

[51] Nimrod Megiddo. Partitioning with two lines in the plane. *Journal of Algorithms*, 6(3):430–433, 1985. doi:10.1016/0196-6774(85)90011-2.

[52] Nimrod Megiddo and Christos H. Papadimitriou. On Total Functions, Existence Theorems and Computational Complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

[53] Frédéric Meunier. Discrete Splittings of the Necklace. *Mathematics of Operations Research*, 33(3):678–688, 2008. URL: http://www.jstor.org/stable/25151879.

[54] Frédéric Meunier and András Sebő. Paintshop, odd cycles and necklace splitting. *Discrete Applied Mathematics*, 157(4):780–793, 2009. doi:10.1016/j.dam.2008.06.017.

[55] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

[56] Svatopluk Poljak and Daniel Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics*, 58(1):99–104, 1986. doi:10.1016/0012-365X(86)90192-5.

[57] Alon Rosen, Gil Segev, and Ido Shahaf. Can PPAD Hardness be Based on Standard Cryptographic Assumptions? *Journal of Cryptology*, 34(1), 2021. doi:10.1007/s00145-020-09369-6.

[58] Patrick Schnider. Ham-Sandwich Cuts and Center Transversals in Subspaces. *Discrete & Computational Geometry*, 64(4):1192–1209, 2020. doi:10.1007/s00454-020-00196-x.

[59] Patrick Schnider. The Complexity of Sharing a Pizza. *Computing in Geometry and Topology*, 1(1):4:1–4:19, 2022. doi:10.57717/cgt.v1i1.5.

[60] Erel Segal-Halevi, Shmuel Nitzan, Avinatan Hassidim, and Yonatan Aumann. Envy-Free Division of Land. *Mathematics of Operations Research*, 45(3):896–922, 2020. doi:10.1287/moor.2019.1016.

[61] Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical Social Sciences*, 45(1):15–25, 2003. doi:10.1016/S0165-4896(02)00087-2.

[62] William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete & Computational Geometry*, 44(3):535–545, 2010. doi:10.1007/s00454-009-9225-8.

[63] Arthur H. Stone and John W. Tukey. Generalized "sandwich" theorems. *Duke Mathematical Journal*, 9(2):356 – 359, 1942. doi:10.1215/S0012-7094-42-00925-6.

[64] Albert W. Tucker. Some topological properties of disk and sphere. In *Proceedings of the first Canadian mathematical congress, Montreal, 1945*, Toronto, 1946. The University of Toronto Press.

[65] Mihalis Yannakakis. Equilibria, fixed points, and complexity classes. *Computer Science Review*, 3(2):71–85, 2009. doi:10.1016/j.cosrev.2009.03.004.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

- (●) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Unfair Necklace Splitting: Algorithmic and Hardness Results

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Stalder	Linus

With my signature I confirm the following:

- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Entlebuch, 29. August 2024	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard