Unfairly Splitting Separable Necklaces

Patrick Schnider
Department of Computer Science, ETH Zürich, Switzerland

Linus Stalder
Department of Computer Science, ETH Zürich, Switzerland

Simon Weber
Department of Computer Science, ETH Zürich, Switzerland

Abstract

The Necklace Splitting problem is a classical problem in combinatorics that has been intensively studied both from a combinatorial and a computational point of view. It is well-known that the Necklace Splitting problem reduces to the discrete Ham Sandwich problem. This reduction was crucial in the proof of PPA-completeness of the Ham Sandwich problem. Recently, Borzechowski, Schnider and Weber [ISAAC'23] introduced a variant of Necklace Splitting that similarly reduces to the α-Ham Sandwich problem, which lies in the complexity class UEOPL but is not known to be complete. To make this reduction work, the input necklace is guaranteed to be *n-separable*. They showed that these necklaces can be fairly split in polynomial time and thus this subproblem cannot be used to prove UEOPL-hardness for α-Ham Sandwich. We consider the more general *unfair* necklace splitting problem on n-separable necklaces, i.e., the problem of splitting these necklaces such that each thief gets a desired fraction of each type of jewels. This more general problem is the natural necklace-splitting-type version of α-Ham Sandwich, and its complexity status is one of the main open questions posed by Borzechowski, Schnider and Weber. We show that the unfair splitting problem is also polynomial-time solvable, and can thus also not be used to show UEOPL-hardness for α-Ham Sandwich.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems; Theory of computation → Computational geometry

Keywords and phrases Necklace splitting, n-separability, well-separation, Ham Sandwich, alpha-Ham Sandwich, unfair splitting, fair division

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.71

Related Version *Full Version*: https://arXiv.org/abs/2408.17126

Funding *Simon Weber*: Swiss National Science Foundation under project no. 204320.

1 Introduction

One of the most famous theorems in fair division is the *Ham Sandwich theorem* [23]. It states that any d point sets in $\mathbb{R}^d$ can be simultaneously *bisected* by a single hyperplane. The Ham Sandwich theorem is closely related to another fair division theorem that lives in $\mathbb{R}$; the *Necklace Splitting theorem*. It states that given n point sets in $\mathbb{R}$ (a *necklace* with n types of *jewels*), we can split the real number line at n points such that when we partition the resulting pieces alternatingly, each of the two parts contains exactly half of the jewels of each type. In fact, the Necklace Splitting theorem can be proven by lifting the necklace to the *moment curve* in $\mathbb{R}^n$, which is the curve parameterised by $(t, t^2, t^3, \dots, t^n)$, and then applying the Ham Sandwich theorem.

Under some additional assumptions on the input points, the Ham Sandwich theorem can be significantly strengthened: If the input point sets are *well-separated* and in general position, the *α-Ham Sandwich theorem* [22] says that we can not only simultaneously *bisect* each point set, but we can for each i choose any number $1 \leq \alpha_i \leq |P_i|$, and find a single


hyperplane that cuts off exactly α_i points from each point set P_i. Informally, a family of point sets is well-separated if the union of any subfamily can be separated from the union of the complement subfamily by a single hyperplane. Borzechowski, Schnider and Weber [6] introduced an analogue of this well-separation condition for necklaces: A necklace is *n-separable*, if any subfamily can be separated from the complement subfamily by splitting the necklace at at most n points. It is shown in [6] that a necklace is n-separable if and only if its lifting to the moment curve is well-separated.

Existence theorems such as the Ham Sandwich and the Necklace Splitting theorem can also be viewed from the lens of computational complexity. The corresponding problems are *total search problems*, i.e., problems in which a solution is always guaranteed to exist, but the task is to actually find a solution. In this setting, the strategy used above to prove the Necklace Splitting theorem using the Ham Sandwich theorem also yields a *reduction* from the Necklace Splitting problem to the Ham Sandwich problem. This reduction was crucial for establishing the PPA-hardness of the Ham Sandwich problem, which is since known to be PPA-complete [12]. The α-Ham Sandwich problem is known to lie in the subclass UEOPL $\subseteq$ PPA [8], but no matching hardness result is known. It is thus natural to ask whether UEOPL-hardness of α-Ham Sandwich could be proven by reduction from a Necklace Splitting problem on n-separable necklaces. Borzechowski, Schnider and Weber [6] showed that the classical (*fair*) Necklace Splitting problem on n-separable necklaces is polynomial-time solvable and thus very unlikely to be UEOPL-hard. However, the natural necklace-splitting-type analogue of α-Ham Sandwich would actually allow for *unfair* splittings on n-separable necklaces, where the first of the two thieves should get exactly α_i jewels of type i, for some input vector α. We settle the complexity status of this problem variant by providing a polynomial-time algorithm. This completely disqualifies Necklace Splitting variants on n-separable necklaces as possible problems to prove UEOPL-hardness of α-Ham Sandwich.

We would like to note that necklace splitting and its extensions to higher dimensions and larger numbers of thieves, as well as other related problems have enjoyed much research interest [1, 2, 4, 5, 7, 10, 11, 13, 14, 17, 20, 21].

1.1 Results

Before we can state our results we have to begin with the most crucial definitions.

▶ **Definition 1** (Necklace). *A* necklace *is a family $C = \{C_1, \ldots, C_n\}$ of disjoint finite point sets in $\mathbb{R}$. The sets C_i are called* colours*, and each point $p \in C_i$ is called a* bead *of colour i.*

We align the following definition of α-cuts in necklaces as closely as possible with the definition of α-cuts in the α-Ham Sandwich theorem (which we will define later). We require that an α-cut of a necklace splits the necklace at exactly one bead per colour, called the *cut point*. Furthermore, the parity of the permutation of how these cut points appear along $\mathbb{R}$ determines whether the first part of the necklace is given to the first or the second thief. Let us make this definition more formal.

▶ **Definition 2** (α-Cut). *Let $C = \{C_1, \ldots, C_n\}$ be a necklace. A set of n cut points $s_1, \ldots, s_n$ such that for all i we have $s_i \in C_i$ defines the subset C^+ of $C_1 \cup \ldots \cup C_n$ that is assigned to the first thief as follows. We first add the points $s_0 := -\infty$ and $s_{n+1} := \infty$, and let $\pi : \{0, \ldots, n+1\} \to \{0, \ldots, n+1\}$ be the permutation such that $s_{\pi(0)} < \ldots < s_{\pi(n+1)}$. We then get the two sets of closed intervals $A_{even} := \{[s_{\pi(i)}, s_{\pi(i+1)}] \mid 0 \leq i \leq n, i \text{ even}\}$ and $A_{odd} := \{[s_{\pi(i)}, s_{\pi(i+1)}] \mid 0 \leq i \leq n, i \text{ odd}\}$. Let A^+ be $A_{sgn(\pi)}$, i.e., the set of intervals corresponding to the parity of π. Then, $\{s_1, \ldots, s_n\}$ is called an* α-cut *of C for the vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, where $\alpha_i = |A^+ \cap C_i|$.*

Definition 2 is illustrated in Figure 1.

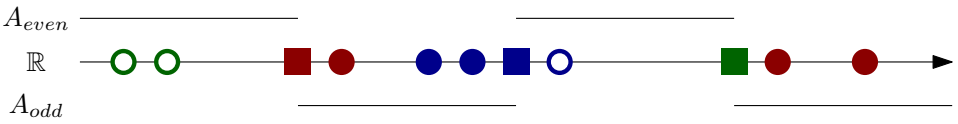

Figure 1 A 3-separable necklace with three colours C_1 (red), C_2 (green), and C_3 (blue). In each colour, the point illustrated as a square is chosen as the cut point s_i. This defines the intervals A_{even} illustrated above the necklace, and the intervals A_{odd} illustrated below. The cut points are coloured 1 3 2 when ordered increasingly. Since the parity of the permutation 0 1 3 2 4 is odd, $A^+ = A_{odd}$. The filled points are thus exactly those in A^+, and A^- consists of the square cut points and the empty circles. Thus this cut is (the unique) $(4, 1, 3)$-cut.

With this definition, n-separability of the necklace and the α-Ham Sandwich theorem guarantee a unique α-cut for every vector α fulfilling $1 \leq \alpha_i \leq |C_i|$, as we will see later. We are now ready to define the α-Necklace-Splitting problem.

▶ **Definition 3** (α-Necklace-Splitting)**.** *Given an n-separable necklace C with n colours, and a vector $\alpha = (\alpha_1, \dots, \alpha_n)$ with $1 \leq \alpha_i \leq |C_i|$, the α-Necklace-Splitting problem is to find the unique α-cut of C.*

We are now ready to introduce our results.

▶ **Theorem 4.** *α-Necklace-Splitting is polynomial-time solvable.*

We contrast this result by showing that without the promise of n-separability, the associated decision problem is NP-complete.

▶ **Theorem 5.** *Given a necklace C with n colours, and a vector $\alpha = (\alpha_1, \dots, \alpha_n)$ with $1 \leq \alpha_i \leq |C_i|$, the problem of deciding whether C has an α-cut is* NP*-complete.*

Note that Borzechowski, Schnider and Weber [6] proved that it is possible to check whether a given necklace is n-separable in polynomial time. This is in contrast to the Ham Sandwich setting, where it has been shown that checking well-separation is co-NP-complete [3].

1.2 Proof Techniques

The algorithm of Borzechowski, Schnider and Weber [6] relies on the following core observation: If some colour only appears in the necklace in two *components* (i.e., consecutive intervals of $\mathbb{R}$ that contain only points of this colour), the smaller of the two components can be discarded since the cut point of that colour must lie in the larger component. While for fair splitting this follows quite immediately – the larger component must be split, otherwise the cut cannot be fair – this observation does not generalise to unfair splitting. We thus have to use a more complicated approach.

We first show that under the promise of n-separability the number of colours that consist of more than two consecutive components is bounded by a constant. For each of these colours we can guess in which component the cut point of this colour lies. For each guess we reduce each of these colours to the single component containing the cut point, and try to compute the α-cut for the resulting necklace. For one of these guesses, the resulting α-cut must be adaptable to an α-cut of the original necklace. To compute the α-cut for these necklaces (in which now every colour consists of at most two components), we reduce the necklace further according to some reduction rules similar to those in [6]. This process will eventually come

to an end; we will reach an *irreducible* necklace. Here comes the crucial part of our proof: We will show that for each n, the irreducible necklaces with n colours all have the same *walk graph* (as introduced in [6] and below in Section 2). We translate α-Necklace-Splitting on irreducible necklaces into an integer linear program (ILP), and use the rigid structure of irreducible necklaces to show that the *primal graph* of this ILP has constant treewidth. Using the FPT algorithm of Jansen and Kratsch [15], we can thus solve these ILPs efficiently.

For the NP-hardness proof of the decision version of α-Necklace-Splitting we use a reduction from e3-Sat.

2 Preliminaries

Let us first formally introduce separability, as defined by Borzechowski, Schnider and Weber [6].

▶ **Definition 6** (Separability). *A necklace C is* k-separable *if for all $A \subseteq C$ there exist k* separator points *$s_1 < \ldots < s_k \in \mathbb{R}$ that separate A from $C \setminus A$. More formally, if we alternatingly label the intervals $(-\infty, s_1], [s_1, s_2], \ldots, [s_{k-1}, s_k], [s_k, \infty)$ with A and $\overline{A}$ (starting with either A or $\overline{A}$), for every interval I labelled A we have $I \cap \bigcup_{c \in (C \setminus A)} c = \emptyset$ and for every interval I' labelled $\overline{A}$ we have $I' \cap \bigcup_{c \in A} c = \emptyset$.*

The separability *$sep(C)$ of a necklace C is the minimum integer $k \geq 0$ such that C is k-separable.*

We call each maximal set of consecutive points that have the same colour c a *component* of c. We say a colour c is an *interval*, if it consists of exactly one component. In other words, a colour c is an interval if its convex hull does not intersect any other colour c'.

Borzechowski, Schnider and Weber further showed that n-separability is strongly related to the notion of *well-separation*.

▶ **Definition 7.** *Let $P_1, \ldots, P_k \subset \mathbb{R}^d$ be point sets. They are* well-separated *if and only if for every non-empty index set $I \subset [k]$, the convex hulls of the two disjoint subfamilies $\bigcup_{i \in I} P_i$ and $\bigcup_{i \in [k] \setminus I} P_i$ can be separated by a hyperplane.*

▶ **Lemma 8** ([6]). *Let C be a set of n colours in $\mathbb{R}$. Let C' be the set of subsets of $\mathbb{R}^n$ obtained by lifting each point in each colour of C to the n-dimensional moment curve using the function $f(t) = (t, t^2, \ldots, t^n)$. Then the set C is n-separable if and only if C' is well-separated.*

The following theorem due to Steiger and Zhao [22] shows that we can always unfairly bisect well-separated point sets.

▶ **Lemma 9** (α-Ham-Sandwich Theorem, [22]). *Let $P_1, \ldots, P_n \subset \mathbb{R}^n$ be finite well-separated point sets in general position, and let $\alpha_1, \ldots, \alpha_n$ be positive integers with $\alpha_i \leq |P_i|$, then there exists a unique $(\alpha_1, \ldots, \alpha_n)$-cut, i.e., a hyperplane H that contains a point from each colour and such that for the closed positive halfspace H^+ bounded by H we have $|H^+ \cap P_i| = \alpha_i$. Here, the positive side of a hyperplane H containing one point p_i per point set P_i is determined by the orientation of these p_i, i.e., for any point $h \in H^+$ the simplex $(p_1, \ldots, p_n, h)$ is oriented positively.*

Note that the $(\alpha_1, \ldots, \alpha_n)$-cut in Lemma 9 is defined by one point from each colour, and the positive side of the cut is determined by the orientation of these points on the hyperplane. This is the motivation of the similar restrictions in our definition of an α-cut in α-Necklace-Splitting (Definition 2).

Through the classical reduction of Necklace Splitting to the Ham-Sandwich problem obtained by lifting the points to the moment curve, as it appeared in many works before [9, 12, 16, 19], we can easily obtain the following theorem.

▶ **Theorem 10.** *α-Necklace-Splitting always has a unique solution.*

Proof (sketch). By Lemma 8, the point sets lifted to the moment curve are well-separated, and thus Lemma 9 applies. α-Necklace-Splitting thus always has a solution, and uniqueness follows from the fact that two different solutions of α-Necklace-Splitting would lift to different solutions of α-Ham Sandwich. ◀

To argue about the separability of necklaces, we use the view of *walk graphs* that were also introduced in [6].

▶ **Definition 11** (Walk graph). *Given a necklace C, the walk graph G_C is the multigraph with $V = C$ and with every potential edge $\{a, b\} \in \binom{V}{2}$ being present with the multiplicity equal to the number of pairs of points $p \in a, p' \in b$ that are neighbouring.*

Note that given a necklace C as a set of point sets, the walk graph can be built in linear time in the size of the necklace $N := \sum_{c \in C} |c|$.

Recall that a graph is *Eulerian* if it contains a Eulerian tour, a closed walk that uses all edges exactly once. A graph is *semi-Eulerian* if it contains a Eulerian path, a (not necessarily closed) walk that uses all edges exactly once.

▶ **Observation 12.** *The walk graph of a necklace is connected and semi-Eulerian, and thus at most two vertices have odd degree.*

The separability of a necklace turns out to be equivalent to the max-cut in its walk graph.

▶ **Definition 13** (Cut). *In a (multi-)graph G on the vertices V, a* cut *is a subset $A \subseteq V$. The* size $\mu(A)$ of a cut *A is the number of edges $\{u, v\}$ in G such that $u \in A$ and $v \notin A$. The* max-cut*, denoted by $\mu(G)$, is the largest size of any cut $A \subseteq V$.*

▶ **Lemma 14** ([6]). *For every necklace C, we have $sep(C) = \mu(G_C)$.*

3 Tractability of the Search Problem

In this section we prove Theorem 4 by providing a polynomial-time algorithm for α-Necklace-Splitting. As mentioned above, the algorithm works in two main phases.

In the first phase, the necklace is first reduced to a necklace where each colour consists of at most two components by guessing the correct component to cut for colours with three or more components. The necklace with only colours with at most two components is then further reduced to an *irreducible* necklace.

In the second phase, we reduce necklace splitting in an irreducible necklace to a labelling problem of its walk graph. This labelling problem is then modelled as an integer linear program, which turns out to be tractable in polynomial time. To prove that this ILP is tractable, we prove some strong structural properties about the walk graphs of irreducible necklaces.

3.1 Reducing Necklaces

Instead of solving α-Necklace-Splitting, we will actually solve the following slightly more general problem. Given a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, we define the complement vector $\overline{\alpha}$ as the vector $(|C_1| - \alpha_1 + 1, \ldots, |C_n| - \alpha_n + 1)$. If α denotes the number of points per point set on the positive side of a cut, $\overline{\alpha}$ denotes the number of points on the other side, both sides including the cut points. Since the cut parity can change in our reduction steps and thus the positive side may become the negative side and vice versa, the following problem is nicer to solve recursively than α-Necklace-Splitting.

▶ **Definition 15.** *α-$\overline{\alpha}$-Necklace-Splitting*

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

3.1.1 Reducing to at most two components per colour

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 16.** *α-$\overline{\alpha}$-Necklace-Splitting$_2$*

Input: *An n-separable necklace $C = \{C_1, \ldots, C_n\}$, where each colour has 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, proven in the full version of this paper.

▶ **Lemma 17.** *Let C be an n-separable necklace with n colours for $n \geq 8$. 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, and at most two colours have more than two components.*

The proof of Lemma 17 as well as the proofs of many other structural results on n-separable necklaces in this paper and in [6] rely on Lemma 14 and the following bound that is a corollary of a theorem of Poljak and Turzík.

▶ **Corollary 18** ([18])**.** *A connected (multi-)graph G with n vertices and m edges has a maximum cut $\mu(G)$ of at least $\omega(G) := \frac{m}{2} + \frac{n-1}{4}$.*

This corollary directly gives a bound on the number of edges in the walk graph of an n-separable necklace. By this we can also bound the degree distribution of the vertices in the walk graph and therefore also the distribution of the numbers of components of colours in the necklace.

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 point. 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 at most two colours have more than two components. The strategy will be to look at these colours and test out every possible component per colour. Again by condition (ii), this requires at most

$4 \cdot 4 = 16 \in O(1)$ tests. That is, for each colour with more than two components we fix a component and remove all 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.

Note that for our recursive calls to be valid, we also need that removing neighbouring intervals yields a $(n-2)$-separable necklace on $n-2$ colours, and removing all but one component from a colour does not destroy n-separability either. Both of these facts are shown in [6, Lemmas 19 and 20].

This lets us conclude the following proposition.

▶ **Proposition 19.** *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 describe an algorithm for α-$\overline{\alpha}$-Necklace-Splitting in this proof, pseudocode of this algorithm can be found in the full version of this paper.

As a base case, for n small enough ($n < 8$) we simply 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 a 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 cuts 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 $(c_{i_1}, \dots, c_{i_{|C^3|}})$ 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 vector might be invalid for this necklace. We therefore use a dummy α value of 1 for these colours. 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, and in the case when removing the components from colours with at least three components.

We first consider the case when removing neighbouring intervals. Note that inserting cuts in neighbouring intervals either changes the parity of the cut permutation for all cuts or for 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 points in the removed intervals, the obtained cuts are indeed valid α- and $\overline{\alpha}$-cuts. This shows that in this case the algorithm returns the unique α- and $\overline{\alpha}$-cuts.

Next we show that the cut returned for the case when there are colours with at least three components 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 lies in. Therefore, there must be one iteration for exactly that choice of components. For that iteration, the cut obtained by the call of the algorithm for α-$\overline{\alpha}$-Necklace-Splitting$_2$ 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 α-cut in C' can be shifted to an α-cut in C. 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. Hence, the cut obtained in this case 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 17 we have $|C^3| \leq 2$ and each colour in C^3 has at most four components. Therefore, the number of iterations over components of C^3 is constant, where each iteration takes time $\mathcal{O}(T_2(n, N) + N)$.

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

3.1.2 Further reductions until irreducibility

To solve the α-$\overline{\alpha}$-Necklace-Splitting$_2$ problem we perform some further reductions to reach a necklace that we cannot further reduce using any techniques known to us, which we will call irreducible.

▶ **Definition 20.** *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 component are of different colours.*

We thus reduce α-$\overline{\alpha}$-Necklace-Splitting$_2$ to the following subproblem.

▶ **Definition 21.** *α-$\overline{\alpha}$-Irr-Necklace-Splitting*

Input: *An n-separable 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.*

To perform this reduction, we search for violations of any of the conditions (ii) to (iv) in Definition 20. Note that although we already removed all neighbouring intervals in Section 3.1.1, neighbouring intervals may have been reintroduced by removing some components from the necklace when removing components from colours, or may be reintroduced now when dealing with any of the other three violations in the further reduction.

In the following we show how the α-cut can be computed for each of the violations to conditions (ii) to (iv). We already know how to deal with violations to condition (ii), i.e., neighbouring intervals, from the previous subsection.

If condition (iii) is violated because 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 either the α-cut or the $\overline{\alpha}$-cut in C' and add the cut the first component at the correct bead. Note that if the first component is of colour C_i and the number of colours $C_j \in C'$ such that $j < i$ is odd, then augmenting the cut will switch the parity of the cut permutation, since adding the first cut in $C_i \notin C'$ adds an odd number of inversions to the permutation. Moreover, adding a cut in the first component flips the positive and the negative side of the rest of the necklace once more. Thus, if the first component is such that the cut permutation parity does *not* flip, we extend the $\overline{\alpha}$-cut of C', and otherwise the α-cut.

The same idea is applied if the last component is an interval C_i. 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 additionally flip the positive and negative side of the rest of the necklace, but the permutation flips parity if the number of colours $C_j \in C'$ such that $j > i$ is odd.

It is easy to see that removing an interval at the beginning or the end of the necklace decreases the separability by 1, since this interval can always be put on the correct side of a cut so one cut is needed to separate it from the rest. Thus, C' is indeed $(n-1)$-separable, and the recursive call is legal.

If condition (iv) is violated because the first and last component are of the same colour, we use a similar idea. We obtain the necklace C' by removing this colour. Note again that C' is $(n-1)$-separable. We again use recursion to get an α- and $\overline{\alpha}$-cut for the remaining colours in C'. The α-cut in C can now be found by trying to augment both the α-cut and the $\overline{\alpha}$-cut of C' by inserting a cut in the first or last component. At least one of the two augmentations must succeed as removing the cut in C_i from the unique α-cut in C must yield either the α- or the $\overline{\alpha}$-cut of C'. Similarly, we can also find the $\overline{\alpha}$-cut of C.

For the sake of completeness we give pseudocode for the way these reductions can be implemented in the full version of this paper. From the arguments above it follows that the algorithm 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 22.** *Let $T_{irr}(n, N)$ be the time to solve α-$\overline{\alpha}$-Irr-Necklace-Splitting on an n-separable 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.*

3.2 Structure of Irreducible Necklaces

To be able to solve α-$\overline{\alpha}$-Irr-Necklace-Splitting, we will first analyse the structure of the walk graphs of irreducible necklaces.

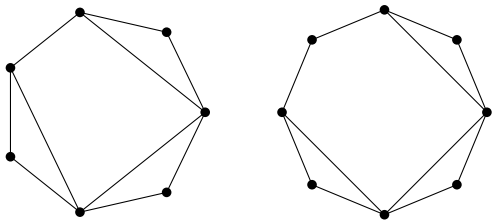

Figure 2 The graphs N_7 to the left and N_8 to the right.

First, we claim that the walk graphs $G = (V, E)$ of irreducible necklaces with n colours can be characterised as follows. A proof of this can be found in the full version of this paper, but the lemma follows quite straightforwardly from Definition 20.

▶ **Lemma 23.** *A graph $G = (V, E)$ on n vertices is the walk graph of an irreducible necklace with n colours if and only if it satisfies all of the following conditions.*

a) *G is semi-Eulerian but not Eulerian,*
b) *for all vertices $v \in V$ we have $\deg(v) \in \{2, 3, 4\}$,*
c) *there are no adjacent vertices of degree 2,*
d) *the maximum cut of G is at most $\mu(G) \leq n$.*

If a graph G is the walk graph for some irreducible necklace, we call G an irreducible walk graph. We will see that all irreducible walk graphs on n vertices are isomorphic. In particular, we claim that the walk graph is isomorphic to the graph N_n defined as follows.

▶ **Definition 24.** *The cycle graph C_n is the cycle on the vertex set $[n]$. The graph P^{odd}_{n-1} is the graph with vertex set $[n]$ and edge set*

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

that is P^{odd}_{n-1} is the graph on vertex set $[n]$ with a path of length $\lfloor (n-1)/2 \rfloor$ starting at vertex 1 and skipping every other vertex. Now the graph N_n obtained by taking the union of the edges in C_n and in P^{odd}_{n-1} is called the irreducible necklace graph *of size n.*

See Figure 2 for two examples of these graphs. Solving α-$\overline{\alpha}$-Irr-Necklace-Splitting heavily relies on the following proposition.

▶ **Proposition 25.** *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 further design of the algorithm. We thus only present the proof in the full version of this paper.

3.3 Splitting Irreducible Necklaces

As mentioned above, we will solve α-$\overline{\alpha}$-Irr-Necklace-Splitting by formulating it as an edge labelling problem in the walk graph, which 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.

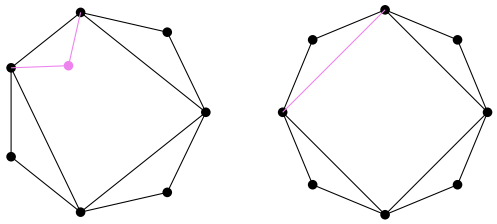

Figure 3 Turning the walk graph to the label graph for $n = 7$ (left) and $n = 8$ (right). The coloured edges and vertices are added to the walk graph to obtain the label graph.

3.3.1 The cut labelling problem

Recall that the edges of the walk graph correspond to intervals between beads of the necklace – where one of the beads is the last bead of one component and the other is the first bead of a component of some different colour. Any choice of one component per colour to put a cut point puts some of these intervals on the positive side of the cut, and others on the negative side. In this way, such a choice of components induces a labelling of the edges of the walk graph by "positive" and "negative". Of course, not every labelling corresponds to a cut. We will now collect some properties of labellings that do correspond to cuts.

For every component of a colour c there are two edges (except if the component appears at the beginning or at the end of the necklace): one edge corresponding to the change of colours 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. The two edges of a traversal are labelled the same if and only if the corresponding component is not chosen to contain a cut point. Thus, if we now consider a vertex corresponding to an interval, it must have exactly one positively labelled incident edge, and one negatively labelled incident edge. Since a colour has exactly one component with a cut point, a bicomponent that is neither the first nor the last colour of the necklace must have either one positive and three negative, or one negative and three positive incident edges.

The edges of the walk graph only contain information about the intervals between components of the necklace. However, we additionally know that 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 is an even number of cut points. Similarly for n odd, the two intervals are on opposite sides of the cut. To capture this information in the edge labelling, we slightly modify our walk graph. We add an edge between the vertices corresponding to the first and last component if n is even, or a subdivided edge (with a new degree two vertex) between these two vertices if n is odd. This newly obtained graph is called the *label graph* (see Figure 3). We see that a choice of one component per colour also induces a labelling of these newly added edges. When n is odd, we see that the two edges incident to the newly introduced vertex v must have different labels, just like if v was a regular vertex corresponding to an interval.

So far, all of our properties are invariant under flipping the labelling of all edges. However, by the cut permutation, any choice of one component per colour fixes the positive side of the cut. We can see that for every labelling fulfilling the previous properties, exactly one of the labelling and its inverse are actually the labelling induced by some cut.

We thus have now found a characterisation of the labellings that are induced by choices of one component per colour to cut. However, we are interested only in α-cuts. We thus would now like to characterise the labellings that are induced by α-cuts.

Clearly, if both edges of a traversal (e, e') are labelled negative, all beads of the corresponding component c of colour C_v must lie on the negative side of the cut. This cut can only be an α-cut if α_c is low enough, i.e., $\alpha_v \leq |C_v| - |c|$. Similarly, if both edges were labelled positive, we would have to have $\alpha_v \geq |c| + 1$. It turns out that if an edge labelling of the label graph fulfils this condition for some fixed α, we can actually place the cut points in the corresponding components to get an α-cut. We thus define the following problem.

For notational convenience we 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 this traversal corresponds to the i-th component of that colour. In our case $e(v, i, j)$ is only defined for $i \in \{1, 2\}$ for every vertex; for intervals only for $i = 1$. We furthermore define $w_v(i)$ as the size of the i-th component of colour v.

▶ **Definition 26.** *We define α-Cut-Labelling as the following problem.*
Input: *The label graph $G = (V, E)$ of some n-separable irreducible necklace $C = \{C_1, \ldots, C_n\}$, and a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{1, \ldots, |C_i|\}$.*
Output: *A subset of the edges $\mathcal{P} \subseteq E$ to be labelled positively (the negatively labelled edges are $\mathcal{N} := E \setminus \mathcal{P}$) such that:*
(1) $\forall v \in V : |\{e \in \mathcal{P} \mid v \in e\}| \in \{1, 3\}$,
(2) $\forall v \in V, i \in \{1, 2\} : \{e(v, i, 1), e(v, i, 2)\} \subseteq \mathcal{P} \implies \alpha_v \geq w_v(i) + 1$,
(3) $\forall v \in V, i \in \{1, 2\} : \{e(v, i, 1), e(v, i, 2)\} \subseteq \mathcal{N} \implies \alpha_v \leq |C_v| - w_v(i)$,
(4) *all edges in $\mathcal{P}$ lie on the positive side of the cut induced by the labelling $(\mathcal{P}, \mathcal{N})$.*

A solution of an α-Cut-Labelling instance for a given α-vector is called an *α-labelling*. The concluding result of this section is the following.

▶ **Proposition 27.** *Let $T_{lab}(n)$ be the time required to solve α-Cut-Labelling on the label graph of any irreducible necklace with n colours. We can solve α-$\overline{\alpha}$-Irr-Necklace-Splitting on any necklace with n colours and N total beads in time $\mathcal{O}(T_{lab}(n, N) + N)$.*

Proof. To solve α-$\overline{\alpha}$-Necklace-Splitting on the necklace C we simply solve α-Cut-Labelling and $\overline{\alpha}$-Cut-Labelling on its label graph and translate the labellings back to α- and $\overline{\alpha}$-cuts. We only consider the α-labelling, since the case for $\overline{\alpha}$ works exactly the same way.

We begin by placing a cut point in the first or last bead of each component that is cut according to the α-labelling. Thanks to conditions (1) and (4) of an α-labelling, this already yields an α'-cut for some other α'. The cut points can now be moved within their components. Moving the cut point of colour v within its component only changes the number of points on the positive side of colour v and leaves other colours unaffected. Moving the cut point from one side of the component to the other allows us to include any number of points of the component on the positive side, from 1 point up to all points. Thanks to conditions (2) and (3), each cut point can be moved such that exactly α_v points of the colour are on the positive side. We have thus reached an α-cut we can return. Since this movement can be computed for each colour individually, it can be performed in $O(N)$. The label graph can also be computed in $O(N)$, and thus the statement follows. ◀

Note that for a given α, there must be a unique α-labelling, since applying the strategy in the proof above to distinct α-labellings would yield distinct α-cuts, a contradiction to the α-Ham Sandwich theorem.

3.3.2 An ILP formulation

In this section we model α-Cut-Labelling as an Integer Linear Program (ILP). To do this, we formulate an ILP such that any solution of the ILP corresponds to a labelling fulfilling conditions (1)–(3) in Definition 26 and vice versa. Condition (4) can then be addressed by observing that for a given instance of α-Cut-Labelling there are only a constant number of labellings fulfilling conditions (1)–(3): namely the labelling induced by the α-cut and the labelling induced by the $\overline{\alpha}$-cut, but with the label of all edges swapped. Instead of merely solving for one solution of the ILP, we will later just enumerate all feasible solutions and check the result against condition (4).

Our ILP formulation will only use binary variables. For the sake of clarity we use **bold** face for variables in our ILP. To model that each edge is either labelled positively or negatively, we introduce two binary variables per edge. For each edge e in the label graph add two binary variables $\mathbf{p}_e$ and $\mathbf{n}_e$ with the interpretation that $\mathbf{p}_e = 1 \iff e$ is labelled positive and $\mathbf{n}_e = 1 \iff e$ is labelled negative. Since any edge must have exactly one label we add the constraint $\mathbf{p}_e + \mathbf{n}_e = 1$ for every edge e.

To model constraints on traversals, we introduce new binary variables $\mathbf{p}_{v,i}$ and $\mathbf{n}_{v,i}$ for every vertex v and possible indices of traversals i. The interpretation of these variables are that in the i-th traversal of v both the edges are labelled both positive or both negative, respectively. To keep these new variables consistent with the variables for edges, we wish to add the constraints $\mathbf{p}_{v,i} = \mathbf{p}_{e(v,i,1)} \cdot \mathbf{p}_{e(v,i,2)}$ and $\mathbf{n}_{v,i} = \mathbf{n}_{e(v,i,1)} \cdot \mathbf{n}_{e(v,i,2)}$. Note that these constraints are not linear, but they can be linearised as follows.

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

We also wish to encode whether the edges corresponding to the i-th traversal have the **s**ame label, no matter what this label is. We thus introduce the variables $\mathbf{s}_{v,i}$ for each vertex v and its traversals i. They are related to $\mathbf{p}_{v,i}$ and $\mathbf{n}_{v,i}$ as follows.

$$\mathbf{s}_{v,i} = \mathbf{p}_{v,i} + \mathbf{n}_{v,i}$$

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

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

Lastly, we need to encode the conditions the α-vector poses on the labelling, that is conditions (2) and (3) of a cut labelling. Note that these conditions are only necessary for vertices with multiple traversals, since for intervals the conditions are always satisfied.

Condition (2) can be implemented using the constraints $\mathbf{p}_{v,i} \cdot w_v(i) + 1 \leq \alpha_v$. For condition (3) we need to be more careful since only using $\mathbf{n}_{v,i} \cdot w_v(i) \geq \alpha_v$ results in a violated condition if $\mathbf{n}_{v,i} = 0$. We thus need to include the case when $\mathbf{n}_{v,i} = 0$ and add a trivial upper bound on α_v. This can be done by using $|C_v|$, the total number of beads for that vertex. We obtain the following constraints for all vertices v with $\deg(v) = 4$.

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

Note that in general ILPs also optimise an objective function. However, we are only interested in the satisfying assignments. Let us now summarise the complete α-Cut-Labelling-ILP.

▶ **Definition 28.** *α-Cut-Labelling-ILP*

Let $G = (V, E)$ be a label graph of some n-separable necklace $C = \{C_1, \ldots, C_n\}$ with colours consisting of at most two components. The α-Cut-Labelling-ILP is the ILP given by the feasibility of the following constraints.

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

By construction, α-Cut-Labelling-ILP is equivalent to conditions (1)–(3) of α-Cut-Labelling in the sense that any solution of the former can be turned to a labelling fulfilling the conditions and vice versa, by simply considering the $\mathbf{p}_e$ and $\mathbf{n}_e$ as the encoding of a labelling.

3.3.3 Solving cut labelling for irreducible necklaces

In this section we show that α-Cut-Labelling-ILP is solvable in polynomial time. To do that we leverage the structure of the ILP given by the fact that the underlying necklace is irreducible. We wish to use the following FPT algorithm by Jansen and Kratsch.

▶ **Theorem 29** ([15]). *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 the ILP only has a constant number of feasible solutions, they can be enumerated in the same time bound.*

The primal graph of an ILP encodes which pairs of variables occur in constraints together. Treewidth is a very well-studied graph parameter that describes how "tree-like" a graph is, with lower treewidth denoting that the graph is more "tree-like". We omit the exact definitions here and refer to the full version of this paper. The following proposition shown in the full version of this paper using a hierarchical approach states that the primal graph of the α-Cut-Labelling-ILP has a constant treewidth. Since the ILP is boolean, the domain is $\mathcal{D} = \{0, 1\}$, and thus from this Theorem 29 gives us polynomial runtime for enumerating all solutions of α-Cut-Labelling-ILP, and thus for solving α-Cut-Labelling.

▶ **Proposition 30.** *Let $C = \{C_1, \dots, C_n\}$ be an irreducible necklace with n colours. Then the primal graph of the α-Cut-Labelling-ILP of that necklace has treewidth at most 55.*

We are now ready to prove the following.

▶ **Proposition 31.** *α-$\overline{\alpha}$-Irr-Necklace-Splitting can be solved in $\mathcal{O}(n \cdot N)$ time.*

Proof. Theorem 29 implies an $\mathcal{O}(n^2)$ algorithm for α-Cut-Labelling-ILP, using that $|\mathcal{D}| = |\{0, 1\}| \in \mathcal{O}(1)$ and $\mathsf{tw}(P(\mathcal{I})) \in \mathcal{O}(1)$ by Proposition 30. 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-Labelling. This check can be implemented in $\mathcal{O}(N)$. This shows that α-Cut-Labelling can be solved in time $\mathcal{O}(n^2 + N)$. Therefore, by Proposition 27 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 Propositions 19 and 22, 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 have thus proven Theorem 4.

4 NP-Completeness of the Decision Problem

In this section we prove Theorem 5, i.e., we show NP-completeness of deciding whether an arbitrary necklace has an α-cut. Since an α-cut can be used as a yes-certificate and verified in polynomial time, this problem is clearly in NP.

Instead of showing NP-hardness of this problem directly, we instead show NP-hardness of the following problem.

▶ **Definition 32.** *$\alpha/\overline{\alpha}$-Necklace-Deciding*

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

$\alpha/\overline{\alpha}$-Necklace-Deciding can be seen as the problem of deciding whether there exist cut points that split the necklace into two sides, one of which has size α, but not necessarily the positive side of the cut defined via the cut parity. Clearly, $\alpha/\overline{\alpha}$-Necklace-Deciding can be solved by testing individually whether C has an α-cut, and whether it has an $\overline{\alpha}$-cut. Thus, the following proposition immediately implies Theorem 5.

▶ **Proposition 33.** *$\alpha/\overline{\alpha}$-Necklace-Deciding is NP-hard.*

Proof. We reduce from E3-SAT. Let $\Phi = C_1 \wedge \cdots \wedge C_m$ be an instance of E3-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- or $\overline{\alpha}$-cut if and only if Φ is satisfiable. As discussed above, we can instead show that Φ is satisfiable if and only if there exist cut points (one per colour) such that *some* chosen side contains the desired number of points α.

To construct ζ, we use the following types of beads. The two types of beads P and N are used to enforce that certain parts of the necklace are on the positive or negative side (we will set $\alpha(P) = |P|$ and $\alpha(N) = 1$). For every variable x we use the types x_0^A and x_0^B to construct a gadget which encodes the truth value of x. For every clause C_i such that $x \in C_i$ we additionally use the types x_i^A and x_i^B to read out the value of x at the clause C_i. To transfer the value of x, we use yet another type of bead x_T. For clauses C_i we only need one more type of bead, which we simply name C_i.

We also need multiple types of beads as *separator beads*. A separator bead only occurs once, and is used to enforce that there is a cut at its location. We denote a separator bead type by S_* where the subscript indicates the part of the necklace the separator bead belongs to.

Finally, we need two types of beads a, b that are only used to enforce the positive side of the cut. The necklace starts with the string $a\,b\,a$ and we set α such that both beads of a must be on the positive side. Since there must be exactly one cut through one bead of a and another cut through the single occurrence of b, the only way of having both a and the b on the positive side is to cut the second occurrence of a, and to put the unbounded interval from $-\infty$ to the first occurrence of a on the positive side of the cut, or to cut through the first occurrence of a, and still putting the unbounded interval from $-\infty$ to the first bead of a on the positive side, which will then also put the second bead of a on the positive side.

The rest of the necklace consists of two main 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 enforce the position of cuts for some types of beads. In the following we describe both parts, starting with the encoding part.

The encoding part first contains the following string for each variable x

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

where k is the number of occurrences of the variable x, that is, the number of clauses C_i such that $x \in C_i$ (either positively or negatively). Then, the encoding part contains the following string for each clause C_i and each variable $x \in C_i$.

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

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

We prove that under the following assumptions the encoding part properly encodes true assignments to Φ. These assumptions will be enforced by the α-vector and the second part of the necklace. We first assume that there is no cut in any bead of types P, x_T, C_i within the encoding part. Furthermore, we assume that a cut is correct if and only if the positive side of the cut *within the encoding part* contains all beads of type P, half of each x_T (note that x_T occurs an even number of times), at most two beads of type C_i, and both beads of types x_i^A and x_i^B (for all i including 0). Recall that a bead at which we cut the necklace is counted towards the positive side.

The only way to make the x_i^A and x_i^B types have both beads on the positive side is by either cutting through the first x_i^A and first x_i^B, or by cutting through the second x_i^A and second x_i^B. The first case will encode the assignment $x =$ "true" and the latter encodes $x =$ "false".

The cuts for $i = 0$ corresponding to a true assignment will move the k beads of type x_T between x_0^A and x_0^B to the negative side. Since we need half of the beads of type x_T on the positive side, this implies that for the cuts in x_i^A and x_i^B for $i \neq 0$, the bead of type x_T must be on the positive side. Hence, the cuts in x_i^A and x_i^B must encode the same assignment as the cuts in x_0^A and x_0^B.

We see that the bead of type C_i between x_i^A and x_i^B is on the negative side of the cut if and only if C_i is satisfied by the variable x. We thus see that at most two of the beads of type C_i are on the positive side if and only if C_i is fulfilled by one of its literals. Hence, we can conclude that under the assumptions listed above, the encoding part correctly encodes correct assignments to Φ.

In the second part of the necklace we will now first enforce that the cut points of type P, x_T, and C_i will lie in this second part.

We add the following three strings. For each clause C_i we add the string

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

where S_i is a new separator bead. Still assuming that P is not getting cut, this enforces that one of the three beads of type C_i is cut.

Then, for each variable x we add the string

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

where again k is the number of clauses containing x, and S_x is a new separator bead. Assuming P and N to not be cut, this enforces the k-th occurrence of x_T to be cut, putting k of the beads of type x_T in this substring on the positive side, and k on the negative side.

Finally, we add the string

$$P\, N.$$

Since 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), we can see that this last bead of type N must be cut: If it was not cut, it would be on the positive side of the cut, since both unbounded intervals must belong to the positive side. However, the cut point of type N is another bead that would be counted to the positive side, violating $\alpha(N) = 1$. If however this last bead of type N is cut, also the last bead of type P must be cut by a symmetric argument.

We thus see that the second part of the string enforces the cuts we assumed in the encoding part. Furthermore, since the second part has exactly half of the beads of type x_T and one up to three beads of type C_i on the positive side, the following α-vector exactly gives us all of the assumptions on the number of beads on the positive side we made in the encoding part.

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

We have argued before that a cut with the correct number of points α on the positive side corresponds to a satisfying assignment of Φ. On the flip side, it is clear that a satisfying assignment can be turned into a cut by cutting the x_i^A, x_i^B at the corresponding places and the C_i at the correct of the three consecutive occurrences, depending on how many literals in C_i are true.

The necklace ζ can be built in polynomial time in m, and we thus get that $\alpha/\overline{\alpha}$-Necklace-Deciding is NP-hard. ◀

For an example of this reduction see the full version of this paper.

References

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

2 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.

3 Helena Bergold, Daniel Bertschinger, Nicolas Grelier, Wolfgang Mulzer, and Patrick Schnider. Well-Separation and Hyperplane Transversals in High Dimensions. In Artur 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.

4 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.

5 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.

6 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.

7 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.

8 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.

9 Jesús De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bulletin of the American Mathematical Society*, 56(3):415–511, 2019. doi:10.1090/bull/1653.

10 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.

11 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.

12 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 638–649, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316334.

13 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.

14 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.

15 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. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48350-3_65.

16 Jiří Matoušek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer New York, 2002. doi:10.1007/978-1-4613-0039-7.

17 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.

18 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.

19 Sambuddha Roy and William Steiger. Some combinatorial and algorithmic applications of the Borsuk–Ulam theorem. *Graphs and Combinatorics*, 23(1):331–341, June 2007. doi:10.1007/s00373-007-0716-1.

20 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.

21 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.

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

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