Tight Bounds for Connected Odd Cycle Transversal Parameterized by Clique-Width

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

Recently, Bojikian and Kratsch [ICALP 2024] presented a novel approach to tackle connectivity problems parameterized by clique-width (cw), based on counting (modulo 2) the number of representations of partial solutions, while allowing for possibly multiple representations to exist for the same partial solution. Using this technique, they got a SETH-tight bound of $\mathcal{O}^*(3^{\text{cw}})$ for the Steiner Tree problem, which was left open by Hegerfeld and Kratsch [ESA 2023]. We use the same technique to solve the Connected Odd Cycle Transversal problem in time $\mathcal{O}^*(12^{\text{cw}})$. Moreover, we prove that our result is tight by providing a SETH-based lower bound excluding algorithms with running time $\mathcal{O}^*((12-\epsilon)^{\text{cw}})$. This answers another question of Hegerfeld and Kratsch [ESA 2023].

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, connected odd cycle transversal, clique-width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.19

Related Version *Full Version*: `https://arxiv.org/abs/2402.08046` [7]

Acknowledgements We are grateful to Vera Chekan for her detailed comments on presentation, and to Falko Hegerfeld for several discussions on the lower bound presented here.

1 Introduction

Parameterized complexity studies the fine-grained complexity of problems by analyzing the dependence of the running time on different measures of the input (called parameters) in addition to its size. Typical parameters are the solution size or structural measures relating, e.g., to sparsity or the existence of certain decompositions of the input. We refer to [12] for an introduction to the field. A central goal is to determine whether a problem with chosen parameter t can be solved in time $f(t) \cdot n^{\mathcal{O}(1)}$ where f is an arbitrary computable function and n is the input size. Such problems are called *fixed-parameter tractable* and form the class FPT. While in general the dependence on t may be arbitrary, there is much interest in making this as low as possible and possibly establishing conditional optimality of obtained time bounds.

The most studied parameter in this direction is the treewidth (of a graph), a measure that sparks interest in different areas of theoretical computer science, not only parameterized complexity. Roughly, the treewidth of a graph G is the smallest value k such that G can be fully decomposed along non-crossing separators of size k. FPT algorithms with single exponential dependence on the treewidth of the graph have long been known for many problems. However, one would still try to improve the exponential dependence on the

Due to space constraints, we defer full proofs and technical details to the full version [7].

parameter, by improving the base of the exponent in the running time. In 2010, Lokshtanov et al. [33, 34] provided the first tight lower bounds for problems parameterized by treewidth, proving that known bases for some benchmark problems cannot be improved assuming the strong exponential time hypothesis (SETH), a widely used hypothesis that, informally speaking, conjectures that the naive solution of the SAT problem by testing all possible assignments is essentially optimal [27, 28]. This has inspired a series of SETH based lower bounds for structural parameters [11, 16, 36, 32, 22, 24].

Connectivity problems like Steiner Tree or Feedback Vertex Set, however, resisted single exponential running time algorithms for a long time. A major obstacle had been keeping track of different connectivity patterns in the boundary of the graph, representing different partial solutions in this graph. This seemed to necessitate a factor of $k^{\mathcal{O}(k)}$ in the running time. In 2011, Cygan et al. [14, 15] showed that this issue can be overcome for many (but not all) connectivity problems, by means of their Cut&Count technique. The core idea is to count the number of connected solutions of a problem modulo 2 in single exponential time by counting the ways that (possibly disconnected) solution candidates can be cut. They used the isolation lemma [35] to solve the underlying decision problem with high probability. Finally, they also proved that many of the resulting bases were tight under SETH.

Relying on the Cut&Count technique, tight bounds for connectivity problems were also derived relative to parameters such as pathwidth [13], cutwidth [6], modular treewidth [25], and clique-width [24, 8]. In this paper, we focus on Clique-width, a graph parameter defined by the smallest value k, such that a given graph can be built recursively from the disjoint union of k-labeled graphs, by allowing to add bicliques between the vertices of two label classes, and to relabel all vertices in a class. Clique-width is a more general parameter than treewidth, since cliques have clique-width at most 2, and unbounded treewidth, while graphs of treewidth k have bounded clique-width (at most $2^{\mathcal{O}(k)}$) [10, 9].

The first single exponential time algorithms for connectivity problems parameterized by clique-width, to the best of our knowledge, were given by Bergougnoux and Kanté [2]. However, the first SETH-tight bounds were given by Hegerfeld and Kratsch [24], where they provided algorithms, based on the Cut&Count technique, and matching SETH-based lower bounds for the Connected Vertex Cover and the Connected Dominating Set problems. They left open however, the tight complexity of other connectivity problems such as Steiner Tree and the Connected Odd Cycle Transversal. A major obstacle towards finding the optimal base of the exponent, as they mention, is a gap between the rank of a matrix defined by the interactions of partial solutions of the underlying problem (called *the compatibility matrix*), and its largest triangular submatrix. Except for a handful examples (Feedback Vertex Set [ctw] [6]), the rank of such matrices has usually been used directly to derive the optimal running time, while current lower bound techniques usually match the size of a largest triangular submatrix.

Very recently Bojikian and Kratsch [8] introduced a new technique called "isolating a representative" closing the gap for the Steiner Tree problem. Their technique is based on a smaller set of possible representations of connectivity patterns. They showed that one can count the number of these representations (modulo 2) in time proportional to the size of a largest triangular submatrix of the corresponding compatibility matrix. They introduced the notion of action-sequences to distinguish different representations of the same solution, and using the isolation lemma to also isolate a unique representation, they were able to solve the underlying decision problem in the same running time, matching known lower bounds.

Our Technique. Following the approach of Bojikian and Kratsch, we provide an optimal (modulo SETH) algorithm for the Connected Odd Cycle Transversal problem parameterized by clique-width with running time $\mathcal{O}^*(12^k)$, proving the following theorem (we refer to Section 2 for definitions):

▶ **Theorem 1.** *There exists a one-sided error Monte-Carlo algorithm (with false negatives only), that given a graph G together with a k-expression of G, and a positive integer $\overline{b}$, solves the* Connected Odd Cycle Transversal *problem in time $\mathcal{O}^*(12^k)$, and outputs the correct answer with probability at least $1/2$.*

Our algorithm counts the number of representations of solutions modulo 2. We achieve this by double representation of partial solutions. The first of which is existential, in the sense that we replace each family of partial solutions at a node of the clique-expression with a smaller family, such that the former completes into a solution in the full graph if and only if the representing family does. In the second step we define an even smaller family, that correctly counts (modulo 2) the number of extensions of the first family into solutions in the rest of the graph. Finally, we assign weights to different actions defined to build the first representing family, and we use the isolation lemma [35] to isolate a single representation with high probability. We make use of fast convolution methods over power lattices [4, 24, 6].

We prove the tightness of our algorithm under SETH by a parameter-preserving reduction from the d-SAT problem. To the best of our knowledge, this is the largest known single exponential base for a natural problem parameterized by a structural parameter. This imposes a challenge when designing the lower bound. As is usual, the lower bound even works for the linear version of the parameter, i.e., for *linear clique-width* [1, 20, 26], and we obtain the following theorem:

▶ **Theorem 2.** *Given a graph G of linear clique-width at most k, assuming SETH, there exists no algorithm that solves the* Connected Odd Cycle Transversal *problem in time $\mathcal{O}^*\big((12-\varepsilon)^k\big)$ even when a linear k-expression of G is provided with G.*

Related work. Clique-width was first studied by Courcelle and Olariu [10] showing that problems expressible in MSO_1 can be solved in FPT time when parameterized by clique-width. However, the resulting algorithms could have a non-elementary dependence on the parameter value k, and hence, are probably not tight for most relevant problems. Espelage et al. [17] provided XP algorithms for some problems not expressible in MSO_1 logic. Nonetheless, some problems were proven to be para-NP-hard when parameterized by clique-width, and hence probably do not admit a polynomial time algorithm on graphs of bounded clique-width [21].

The SETH-tight lower bounds by Lokshtanov et al. [34] were followed by series of tight bounds for structural parameters [6, 36, 19, 30, 32]. Iwata and Yoshida [29] proved that Vertex Cover has the same base parameterized by treewidth and clique-width, providing one of the earliest tight bounds for clique-width. Further tight bounds were obtained for counting perfect matchings, graph coloring and other problems [11, 18, 24, 31, 32].

Finally, we make use of fast convolution methods. These methods are usually used to combine different partial solutions in different parts of the graph along a common boundary set efficiently. Björklund et al. [3] introduced fast convolution over the subset lattice solving the Steiner Tree problem parameterized by the number of terminals more efficiently. Fast subset convolution, and variations thereof have since become a standard technique to handle a join node in dynamic programming algorithms over tree decompositions [37, 15]. Björklund et al. [4] showed that one can compute the so-called join-product over a lattice more efficiently,

if the underlying lattice admits a small number of so-called irreducible elements. Building on this result, Hegerfeld and Kratsch [24] provided a fast convolution technique over power lattices resulting in tight algorithms for connectivity problems parameterized by clique-width.

Organization. In Section 2 we provide preliminaries and introduce some notation. In Section 3 we define connectivity patterns and pattern operations. In Section 4 we describe the dynamic programming algorithm and prove its running time. In Section 5 we define action-sequences, and we use them to prove the correctness of the algorithm. Finally, in Section 6 we present the lower bound. We conclude with final remarks in Section 7.

2 Preliminaries

We denote by $[k]$ the set $\{1, 2, \dots k\}$ and by $[k]_0$ the set $[k] \cup \{0\}$. A *labeled graph* is a graph $G = (V, E)$ together with a *labeling function* $\text{lab}\colon V \to \mathbb{N}$. We usually omit lab and assume that it is given implicitly with G. We define a *clique-expression* μ as a well-formed expression defined by the following operations on labeled graphs:

- *Introduce* vertex $i(v)$ for $i \in \mathbb{N}$: constructs a graph with a single vertex labeled i.
- *Union* $G_1 \uplus G_2$: the disjoint union of G_1 and G_2.
- *Relabel* $\rho_{i\to j}(G)$ for $i \neq j \in \mathbb{N}$: change the label of all vertices labeled i to j.
- *Join* $\eta_{i,j}(G)$ for $i \neq j \in \mathbb{N}$: add all edges between vertices labeled i and vertices labeled j.

We denote the graph resulting from a clique-expression μ by G_μ, and the constructed labeling function by lab_μ. We associate with a clique-expression μ a syntax tree $\mathcal{T}_\mu$ in the natural way, and associate with each node $x \in V(\mathcal{T})$ the corresponding operation. Let $\mathcal{V}_\mu = V(\mathcal{T}_\mu)$. We omit μ when clear from the context. The subtree rooted at each node x induces a subexpression μ_x. We define $G_x = G_{\mu_x}$, $V_x = V(G_x)$, $E_x = E(G_x)$, $\text{lab}_x = \text{lab}_{\mu_x}$, $\mathcal{T}_x = \mathcal{T}_{\mu_x}$, and $\mathcal{V}_x = \mathcal{V}_{\mu_x}$. We also define the set $\mathcal{V}_x^C$ as the set of all *introduce* nodes in $\mathcal{T}_x$, and $\mathcal{V}_x^{CJ}$ as the set of all *introduce and join* nodes in $\mathcal{T}_x$. Given a set $S \subseteq V$, we define $S_x = S \cap V_x$. For a mapping $g\colon S \to U$, we denote by g_x the mapping $g_{|S_x}$, i.e. the restriction of g to $S \cap V_x$.

We call a clique-expression μ *linear*, if it holds for each union node x with children x_1 and x_2 that μ_{x_2} is an introduce operation. We say that G is k-labeled, if it holds that $\text{lab}(v) \leq k$ for all $v \in V(G)$. We say that a clique-expression μ is a *k-expression* if G_x is a k-labeled graph for all $x \in \mathcal{V}$. We define the (linear) clique-width of a graph G, denoted by $\text{cw}(G)$ ($\text{lcw}(G)$ resp.) as the smallest value k such that there exists a (linear) k-expression μ, with G_μ isomorphic to G. We can assume without loss of generality, that any given k-expression defining a graph $G = (V, E)$ uses at most $O(|V|)$ union operations, and at most $O(|V|k^2)$ unary operations [2, 10].

A mapping $g : V \to [q]$, for $q \in \mathbb{N}$, is a *proper q-coloring* of G, iff for all $\{u, v\} \in E$ it holds that $g(u) \neq g(v)$. A graph G is *q-colorable* if it admits a proper q-coloring. A set $S \subseteq V$ is an *odd cycle transversal* of G, if $G \setminus S$ is 2-colorable. In this case, we call a proper 2-coloring g of $G \setminus S$ a *witness*. In the Connected Odd Cycle Transversal problem, given is a graph G and a positive integer $\bar{b}$, asked is whether G admits an odd cycle transversal S of size at most $\bar{b}$, such that S induces a connected subgraph of G.

We denote the symmetric difference by Δ. A *weight function* ω is a mapping from a set U to some ring (usually $\mathbb{Z}$). Given $S \subseteq U$, we define $\omega(S) = \sum_{u\in S} \omega(u)$. Let $f : U^k \to U$ be a mapping (not explicitly defined as a weight function). For $S_1, \dots S_k \subseteq U$, we define $f(S_1, \dots S_k) = \{f(u_1, \dots, u_k)\colon u_i \in S_i\}$. For $S \subseteq U$, we define $f^{-1}(S)$ in a natural way. Finally, we define the operation $\overset{\triangle}{f}\colon (2^U)^k \to 2^U$ with $\overset{\triangle}{f}(S_1, \dots, S_k) = \triangle_{(u_1,\dots,u_k)\in S_1\times\cdots\times S_k} \big\{f(u_1, \dots, u_k)\big\}$, and call it *exclusive* f. We define the union of two mappings $f \dot\cup g$ over disjoint domains in a natural way.

A short introduction to lattices can be found in the full version. For a lattice $(\mathcal{L}, \leq)$, we define the set of *join-irreducible* elements $\mathcal{L}_\vee \subseteq \mathcal{L}$ as the set of elements $u \in \mathcal{L}$ where for all $v, w \in \mathcal{L}$ with $u = v \vee w$ it holds that $u = v$ or $u = w$. We say that a lattice $(\mathcal{L}, \leq)$ is given in the *join representation* if its elements are given as $\mathcal{O}(\log |\mathcal{L}|)$-bit strings, together with the elements of $\mathcal{L}_\vee$, and an algorithm $\mathcal{A}_\mathcal{L}$ that computes the join $u \vee v$ for $u \in \mathcal{L}$ and $v \in \mathcal{L}_\vee$.

Let $A, B \in \mathbb{F}^\mathcal{L}$ be two vectors over a ring $\mathbb{F}$. We define the $\vee$-*product* $A \otimes B \in \mathbb{F}^\mathcal{L}$ where for $z \in \mathcal{L}$ it holds that $A \otimes B[z] = \sum_{\substack{x,y \in \mathcal{L} \\ x \vee y = z}} A[x] \cdot B[y]$. We refer to the full version for details.

Conventions. Along the upper bound, let $G = (V, E)$ be the input graph and let $\overline{b} \in \mathbb{N}$ be the target value in the Connected Odd Cycle Transversal instance. Let v_0 be a fixed vertex that we choose later. Let $n = |V|$ and $m = |E|$. Let μ a k-expression of G for some value $k \in \mathbb{N}$, and $\mathcal{T}$ be the corresponding syntax tree. Let r be the root of $\mathcal{T}$. A *partial solution* at $x \in \mathcal{V}$ is a set of vertices $S \subseteq V_x$, representing the part of the solution introduced so far, such that $G_x \setminus S$ is bipartite.

For a function f, we denote by $\mathcal{O}^*(f(k))$ the class $f(k) \cdot n^{\mathcal{O}(1)}$, where the star hides any polynomial factor of n. We define the ground set $U = [k]$, and call its elements *labels*. Let $U_0 = [k]_0$. Let $W \in \mathbb{N}$ be some fixed value, and $\omega : \mathcal{V} \times [4] \to [W]$ a weight function, both fixed later. We define the intervals $\overline{B} = [n]_0$ and $\overline{W} = [|\mathcal{V}| \cdot W]_0$, and we use the indices b and w to iterate over $\overline{B}$ and $\overline{W}$ respectively. We skip defining these indices repeatedly to avoid redundancy.

3 Connectivity patterns

Patterns are structures that represent connectivity in labeled graphs. We use the definition of patterns and some of their properties from [8]. A *pattern* p is a set of subsets of U_0, such that there exists exactly one set $Z_p \in p$ that contains the element 0 in it. We call this set the *zero-set* of p. We denote by $\mathscr{P}$ the family of all patterns. Let $\text{label}(p) \subseteq U$ be set of labels that appear in at least one set of p, and $\text{singleton}(p) \subseteq U$ the set of labels that appear as singletons in p (excluding the element 0). We define the set of *incomplete labels* $\text{inc}(p) = \text{label}(p) \setminus \text{singleton}(p)$. We denote a pattern $\{\{i_1^1, \ldots, i_{k_1}^1\}, \ldots, \{i_1^\ell, \ldots, i_{k_\ell}^\ell\}\}$ by $\left[i_1^1 \ldots i_{k_1}^1, \cdots, i_1^\ell \ldots i_{k_\ell}^\ell\right]$, but only if this notation does not cause confusion. We also omit the zero-set when it is a singleton.

▶ **Definition 3.** *Given a labeled graph G and a set $S \subseteq V(G)$, we define the pattern* $\text{pat}_G(S)$ *as the sets of labels appearing in each connected component of $G[S]$. Formally, for each connected component C, we add the set* $\text{lab}(C)$ *to the pattern. We add the label* 0 *to the set corresponding to the component that contains v_0, or as singleton if $v_0 \notin S$.*

Such patterns carry enough information about the connectivity of partial solutions. We define pattern operations, that allow us to build connectivity patterns corresponding to different partial solutions recursively over the nodes of $\mathcal{T}$ (see [8] for details).

▶ **Definition 4.** *We define the operations* join, relabel, union, *and* add *over patterns as follows: The union operation of two patterns $p \oplus q$ is defined as the pattern r resulting from the union of p and q by replacing the zero-sets of both patterns by the union of these two sets. The relabel operation ($p_{i \to j}$) changes all occurrences of the label i into the label j. The join operation ($p \sqcup q$) exhaustively merges all sets of p and q that intersect. Finally, we define the add operation $\square_{i,j}(p)$ as $p \sqcup [ij]$ if $\{i, j\} \subseteq \text{label}(p)$, or as p otherwise.*

As we shall see later, these operations allow to build all patterns corresponding to all partial solutions of a fixed size over $\mathcal{T}$. This can be done by deciding at each leaf node, whether to include the introduced vertex in the solution or not. One can then compute these families at each node by applying the corresponding pattern operations on the families at the children nodes of this node. It is not hard to see that a union node corresponds to a union operation, a relabel node corresponds to a relabel operation, and a join node to an add operation. A correctness proof of this is given later implicitly, when proving that the so-called solution-sequences correspond to all partial solutions of a given weight.

Now we define the compatibility relation over patterns. This relation indicates whether two partial solutions in two labeled subgraphs join into a connected graph by connecting all vertices of the same label between them. Formally, we call two patterns p and q *compatible* ($p \sim q$), if their join is a single set.

This notion of compatibility will allow us to define two kinds of representation, an existential one (representation), and a (modulo 2) count-preserving one (parity-representation). Based on these, we identify two special families of patterns, namely the family of *complete patterns* $\mathscr{P}_C$ and the family of *CS-patterns* $\mathscr{P}_{CS}$. We will show later that one can represent each family of patterns by a family of complete patterns, and parity-represent each family of complete patterns by a family of CS-patterns.

▶ **Definition 5.** *A pattern p is* complete *if it holds that* $\text{singleton}(p) = \text{label}(p)$*, i.e. each label that appears in this pattern, appears as a singleton as well. We call a complete pattern a* CS-pattern*, if it consists only of a zero-set and singletons. We denote the family of* complete patterns *by $\mathscr{P}_C$ and the family of* CS-patterns *by $\mathscr{P}_{CS}$.*

▶ **Lemma 6.** *The pattern* $[0]$ *is the only complete pattern compatible with the pattern* $[0]$*.*

We will show that all defined pattern operations preserve (parity-)representation, in the sense that if R represents S, then $\text{Op}(R)$ represents $\text{Op}(S)$ as well. Therefore, we can replace the families of patterns of all partial solutions over $\mathcal{T}$ by representations thereof, which allows us to restrict the dynamic programming algorithm to CS-patterns only, correctly counting (modulo 2) the number of all representations of all solutions. Therefore, the running time will depend on the number of CS-patterns over k labels instead of the number of all patterns.

Finally, we use the isolation lemma to show that, by assigning random weights to the different representations of different partial solutions, we get a unique minimum-weight representation with high probability. This allows to solve the decision problem in the given time. The running time is implied by the number of CS-patterns over k labels, combined with a proper 2-coloring of the remaining part of the graph.

Before we present the algorithm, we still need to define the notion of *actions* and the operation PREP. Actions are series of "forget" and "fix" operations over patterns, that build a representation of a pattern by deciding whether a given label will still be relevant (in the future) for deciding the connectivity of a solution. Using actions, we build an equivalent representations of all partial solutions recursively over $\mathcal{T}$ using complete patterns only. The operation PREP is then used to create a parity-representation thereof, consisting of CS-patterns only.

▶ **Definition 7.** *Given a pattern p and a label $i \in \text{inc}(p)$, we define the* fix *and* forget *operations, where* $\text{fix}(p, i)$ *adds the singleton* $\{i\}$ *to p, and* $\text{forget}(p, i)$ *removes the label i from all sets of p. Now we define actions* $\text{Ac}(p, \ell)$ *for patterns p with* $|\text{inc}(p)| \leq 2$ *and* $\ell \in [4]$*, by either fixing or forgetting the labels in* $\text{inc}(p)$ *independently.*

It was shown in [8] that $\{\text{fix}(p,i), \text{forget}(p,i)\}$ represents p. Hence, by applying the actions $\text{Ac}(p,\ell)$ for $\ell \in [4]$ on a pattern p with $|\operatorname{inc}(p)| \leq 2$, we get a family of complete patterns that represents p. It is not hard to see that applying pattern operations on CS-patterns yields patterns p with $|\operatorname{inc}(p)| \leq 2$. In particular, $\mathscr{P}_{CS}$ is closed under union and relabeling, whereas for $p \in \mathscr{P}_{CS}$ and $q = \square_{i,j} p$, it holds that $\operatorname{inc}(q) \subseteq \{i,j\}$.

▶ **Definition 8.** *Let $p \in \mathscr{P}_C$. We define $\text{PREP}(p) \subseteq \mathscr{P}_{CS}$ as follows: Let $P_0 = \{p\}$, and let $S_1, \dots S_r$ be all non-singleton sets of p different from Z_p. We define P_i for $i \in [r]$ as follows:*

$$P_i = \bigtriangleup_{q \in P_{i-1}} \Big\{ \big(q \setminus \{Z_q, S_i\}\big) \cup \{Z_q \cup S' \} \colon S' \subset S_i \Big\}.$$

Then we define $\text{PREP}(p) = P_r$. Given a set $S \subseteq \mathscr{P}_C$, we define $\text{PREP}(S) = \bigtriangleup_{p \in S} \text{PREP}(p)$.

Note that each pattern of $\text{PREP}(p)$ results from p by removing all non-singleton sets different from Z_p, and adding some subset of their union to Z_p.

4 The Algorithm

Now we present our main result, namely a dynamic programming algorithm over $\mathcal{T}$. We start by defining the family of states $\mathscr{I}$ that we use to index the dynamic programming tables.

CS-Patterns. We use the family $\mathscr{P}_{CS}$, similar to [8], to count the number of weighted representations of partial solutions in G_x for each $x \in \mathcal{V}$. We note that a CS-pattern p is uniquely defined by its set of labels and its zero-set; Given $X, Y \subseteq U$ the set of labels and the zero-set of p respectively with $Y \subseteq X$, we define $p = \big\{\{u\} \colon u \in X\big\} \cup \big\{\{0\} \cup Y\big\}$.

We define the partial ordering $\preceq_{CS}$ over $\mathscr{P}_{CS}$, where for $p, q \in \mathscr{P}_{CS}$ it holds that $p \preceq_{CS} q$ if and only if $Z_p \subseteq Z_q$ and $\text{label}(p) \subseteq \text{label}(q)$. Clearly, $(\mathscr{P}_{CS}, \preceq_{CS})$ defines a lattice, with $p \vee q = r$ where $\text{label}(r) = \text{label}(p) \cup \text{label}(q)$, and $Z_r = Z_p \cup Z_q$. Hence, the join operation over this lattice corresponds to the union operation over CS-patterns $p \oplus q$.

Colorings and bipartition. We define the set $C_0 = \{\boldsymbol{B}, \boldsymbol{W}\}$, and call it the set of *basic colors*, where $\boldsymbol{B}$ stands for black, and $\boldsymbol{W}$ stands for white. We define the set of *colors* $\overline{C} = \{\emptyset, \boldsymbol{B}, \boldsymbol{W}, \boldsymbol{BW}\}$, where each element of $\overline{C}$ corresponds to a different subset of C_0 in the natural way. We define the ordering $\sqsubseteq_c$ over $\overline{C}$ given by inclusion over the corresponding subsets, namely, given by the two chains $\emptyset \sqsubseteq_c \boldsymbol{B} \sqsubseteq_c \boldsymbol{BW}$ and $\emptyset \sqsubseteq_c \boldsymbol{W} \sqsubseteq_c \boldsymbol{BW}$. We denote by $\sqcup$ the join operation, and by $\sqcap$ the meet operation over the underlying lattice. We call two colors $x, y \in \overline{C}$ *compatible* (denoted by $x \sim y$), if $x \sqcap y = \emptyset$.

We call a mapping $c : U \to \overline{C}$ a *coloring*. We denote the family of all colorings of U by $\mathscr{C} = \overline{C}^U$. We define the ordering $\preceq_C$ over colorings, where for $c_1, c_2 \in \mathscr{C}$, it holds that $c_1 \preceq_C c_2$ if it holds that $c_1(i) \sqsubseteq_c c_2(i)$ for all $i \in U$. Finally, we denote by $\oplus_C$ the join operation over the underlying lattice (pointwise join). For $c \in \mathscr{C}$ and $i, j \in [k]$, we define the coloring $c_{i \to j}$ as $c_{i \to j}(j) = c(i) \sqcup c(j)$, $c_{i \to j}(i) = \emptyset$ and $c_{i \to j}(\ell) = c(\ell)$ for $\ell \notin \{i, j\}$. For $i \in [k]$ and $X \in \overline{C}$, let $c_i^X \in \mathscr{C}$ with $c_i^X(i) = X$ and $c_i^X(\ell) = \emptyset$ for $\ell \neq i$.

Dynamic programming tables. We define the family of *states* $\mathscr{I} = \mathscr{P}_{CS} \times \mathscr{C}$ as the set of combinations of a CS-pattern and a coloring. For $(p, c) \in \mathscr{I}$, the pattern p indicates the connectivity of a partial solution, and the coloring c indicates the *basic colors* assigned to the vertices of each label class in a witness of this partial solution. This coloring allows to extend a partial solution, preserving a proper 2-coloring of the rest of the graph.

We define the ordering $\preceq$ over $\mathscr{I}$ with $(p_1, c_1) \preceq (p_2, c_2)$ if and only if $p_1 \preceq_{CS} p_2$ and $c_1 \preceq_C c_2$. It holds that $\preceq$ is the product of two lattices, and hence, it holds that $(p_1, c_1) \vee (p_2, c_2) = (p, c)$, where $p = p_1 \oplus p_2$ and $c = c_1 \oplus_C c_2$.

Description of the algorithm. Now we describe the algorithm. Intuitively, actions (Definition 7) allow to build (existential) representations of partial solutions by a dynamic programming scheme over $\mathcal{T}$ using complete patterns. However, since the number of complete patterns is at least the number of all partitions of U, we seek to reduce the space of the states by representing (in a count-preserving manner) these patterns using CS-patterns only.

We define the vectors $T\big[x, b, w\big] \in \{0,1\}^{\mathscr{I}}$ for $x \in \mathcal{V}, b \in \overline{B}$ and $w \in \overline{W}$, that constitute the dynamic programming tables of our algorithm, and we show that these vectors can be computed in time $\mathcal{O}^*(12^k)$. In the following sections, we will prove the correctness of the algorithm in two stages. In the first stage we show that any pattern, corresponding to a partial solution at a node x, can be represented by a set of complete patterns resulting from applying different actions at each node in $\mathcal{V}_x^{CJ}$. We assign different weights to the actions at each node resulting in different weights for different representations of partial solutions. In the second stage, we show that the tables $T\big[x, b, w\big]$ count for each CS-pattern p, the number of representations of weight w of solutions of size b that are compatible with p. By appropriately choosing the weight function, we show that this suffices to solve the Connected Odd Cycle Transversal problem with high probability.

▶ **Algorithm 1.** *For $x \in V(\mathcal{T})$, and for two values b, w, we define the vectors $T\big[x, b, w\big] \in \{0,1\}^{\mathscr{I}}$ recursively over $\mathcal{T}$ as follows:*

- *Introduce node $\mu_x = i(v)$: Let $p = [0i]$ if $v = v_0$, and $p = [i]$ otherwise. Let $p_1 = \text{forget}(p, i)$ and $p_2 = \text{fix}(p, i)$. We set all values $T\big[x, b, w\big]\big[q, c\big]$ to zero, for all values of q and c, and then we add one to each of the following entries:*

$$T\big[x, 1, \omega(x,1)\big]\big[(p_1, c_i^{\boldsymbol{\emptyset}})\big] \qquad T\big[x, 1, \omega(x,2)\big]\big[(p_2, c_i^{\boldsymbol{\emptyset}})\big]$$
$$T\big[x, 0, \omega(x,3)\big]\big[([0], c_i^{\boldsymbol{B}})\big] \qquad T\big[x, 0, \omega(x,4)\big]\big[([0], c_i^{\boldsymbol{W}})\big],$$

  where the first two entries correspond to adding the vertex to a partial solution and computing a representation thereof, and the latter two entries correspond to the two different partial colorings of this vertex.
- *Relabel node $\mu_x = \rho_{i\to j}(\mu_{x'})$: For each state $(p, c) \in \mathscr{I}$ we define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\substack{(p', c') \in \mathscr{I}, \\ p'_{i\to j} = p \wedge c'_{i\to j} = c}} T\big[x', b, w\big]\big[(p', c')\big].$$

- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$: For $(p, c) \in \mathscr{I}$, if $c(i) \not\sim c(j)$, we set $T\big[x, b, w\big]\big[(p, c)\big] = 0$. Otherwise, we define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\ell \in [4]} \sum_{\substack{p' \in \mathscr{P}_{CS} \\ p \in \text{PREP}\left(\text{Ac}(\square_{i,j} p', \ell)\right)}} T\big[x', b, w - \omega(x, \ell)\big]\big[(p', c)\big].$$

  Intuitively, this corresponds to iterating over all patterns p' at x', applying an add operation to p', and then applying all four actions to the resulting pattern to compute a representation thereof. We add parity-representations of the resulting patterns to $T\big[x, b, w\big]$.
- *Union node $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$: We define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\substack{b_1 + b_2 = b \\ w_1 + w_2 = w}} \sum_{\substack{p_1 \oplus p_2 = p \\ c_1 \oplus_C c_2 = c}} T\big[x_1, b_1, w_1\big]\big[(p_1, c_1)\big] \cdot T\big[x_2, b_2, w_2\big]\big[(p_2, c_2)\big].$$

Now we show how to compute these tables in time $\mathcal{O}^*(12^k)$. We assume that an update of a single entry can be done in logarithmic time in the size of the tables (hence, polynomial in n). Although processing a union node x in the naive way requires time polynomial in $(12^k)^2 = 144^k$, we apply a fast convolution technique to compute these tables more efficiently.

In particular, we use a result from [23], which informally states that the $\vee$-product over powers of a finite lattice can be computed efficiently. In the full version, we define a simple lattice over the set $[3] \times [4]$. We show that the lattice $\mathscr{I}$ is isomorphic to the k-th power of this lattice. It follows by the mentioned result that the $\vee$-product over $\mathscr{I}$ can be computed in time $\mathcal{O}^*(12^k)$ implying the same running time (up to a polynomial factor) to process a union node. The running time of the algorithm is a direct consequence of this result, since the tables at a join or a relabel node can be computed in a straight-forward way by iterating over all entries of the tables at a child node and updating the right indices accordingly.

▶ **Corollary 9.** *All tables $T[x, b, w]$ for all values of x, b, w can be computed in time $\mathcal{O}^*(12^k)$.*

5 Solution representation

Now we introduce solution-sequences and action-sequences. A *solution-sequence* is defined by the actions taken at all introduce nodes in a subtree of $\mathcal{T}$ to create a partial solution (together with a witness thereof). We show a bijective mapping between solution-sequences at a node and partial solutions (combined with witnesses) at that node. An *action-sequence* is defined by the actions taken at both introduce and join nodes in a subtree of $\mathcal{T}$, encoding the forget and fix operations along $\mathcal{T}$. Each action-sequence corresponds to a different representation of a partial solution. Intuitively, the set of all action-sequences extending a solution-sequence build a full complete-pattern representation of the corresponding partial solution.

▶ **Definition 10.** *Given a labeled graph G, a set $S \subseteq V(G)$ and a mapping $g\colon S \to \{\boldsymbol{B}, \boldsymbol{W}\}$, we define the coloring $\mathrm{col}^g_G \in \mathscr{C}$ corresponding to g by assigning to each label $\ell \in [k]$ the color defined as follows: let $b = \boldsymbol{B}$ if $\boldsymbol{B} \in g(\mathrm{lab}^{-1}_{G[S]}(\ell))$, and $\emptyset$ otherwise, and let $w = \boldsymbol{W}$ if $\boldsymbol{W} \in g(\mathrm{lab}^{-1}_{G[S]}(\ell))$, and $\emptyset$ otherwise. We define $\mathrm{col}^g_G(\ell) = b \oplus_C w$.*

Intuitively, col^g_G is the coloring defined by assigning to each label ℓ the join of all basic colors assigned to vertices labeled ℓ in S, or $\emptyset$ if no such vertex exists. Hence, it indicates the colors assigned to each label in a witness of a partial solution.

From now on, we shorthand $\mathrm{pat}_x(S)$ for $\mathrm{pat}_{G_x}(S)$, and col^g_x for $\mathrm{col}^g_{G_x}$.

▶ **Definition 11.** *A* solution-sequence *is a mapping $\pi\colon \mathcal{V}^C_x \to \{0, 3, 4\}$. The cost of π is defined as $b(\pi) = |\pi^{-1}(0)|$. The solution-subsequence π_y induced by a node $y \in \mathcal{V}_x$ is the restriction of π to $\mathcal{V}^C_y$. We define the* pattern *$o_\pi = p$ in a natural way, where for an introduce node, if $\pi(x) = 0$ we define $p = [0i]$ if $v = v_0$, and $p = [i]$ if $v \neq v_0$. We define $p = [0]$ otherwise. For different nodes x, p is then defined by applying the corresponding pattern operation to the patterns o_{π_y} for children nodes y of x. We also define the coloring c_π. For an introduce node $i(v)$, we define $c_\pi = C^{\emptyset}_i$ if $i = 0$, $C^{\boldsymbol{B}}_i$ if $\pi(x) = 3$, and $C^{\boldsymbol{W}}_i$ if $\pi(x) = 4$. For a relabel node, we define $c_\pi = (c_{\pi_y})_{i \to j}$. For a join node, we define $c_\pi = c_{\pi_y}$, and for a union node we define $c_\pi = c_{\pi_{y_1}} \oplus_C c_{\pi_{y_2}}$. We say that π is a* valid *solution-sequence, if for each join node $\mu_y = \eta_{i,j,(\mu_{y'})}$, it holds that $c_{\pi_{y'}}(i)$ and $c_{\pi_{y'}}(j)$ are compatible.*

We define an action-sequence *as a mapping $\tau\colon \mathcal{V}^{CJ} \to [4]$ with the cost of τ defined by $b(\tau) = \big|\{y \in \mathcal{V}^C_x \colon \tau(y) \in \{1, 2\}\}\big|$. The weight of τ is defined by $\omega(\tau) = \sum_{y \in \mathcal{V}^{CJ}_x} \omega(y, \tau(y))$. The action-subsequence τ_y at a descendant y of x is defined by the restriction of τ to $\mathcal{V}^{CJ}_y$. Finally, we define the pattern p_τ in a similar way to o_π, where for an introduce node we*

apply a forget operation if $\tau(x) = 1$, and a fix if $\tau(x) = 2$. For a join node, we follow the add operation by the applying the action $\tau(x)$ on the resulting pattern. The coloring c_τ is defined in the same way as c_π where at an introduce node we define replace $\pi(x) = 0$ by $\tau(x) \in \{1, 2\}$. We define a valid *action-sequence in the same way as for solution-sequences. We say that (o_τ, c_τ) is the* pair *generated by τ.*

The notion of valid sequences ensures the validity of witnesses, as it forbids adding edges between vertices of the same color. Now we show the correspondence between solution-sequences and partial solutions at a node x.

▶ **Definition 12.** *Given a set $S \subseteq V_x$ and a mapping $g\colon B_x \setminus S \to C_0$, we define the solution-sequence $\pi_x^{S,g} = \pi$ where for v the vertex introduced at x we have $\pi(x) = 0$ if $v \in S$, $\pi(x) = 3$ if $g(v) = \boldsymbol{B}$, and $\pi(x) = 4$ if $g(v) = \boldsymbol{W}$.*

This forms a bijective mapping between the family of partial solutions at a node x and the family of all solution-sequences at x. In the full version, we show that $b(\pi) = |S|$, $\mathrm{pat}_x(S) = o_\pi$, $\mathrm{col}_x^g = c_\pi$. We also prove that g is a proper 2-coloring of $G_x[V_x \setminus S]$ if and only if π is valid. Now we show that action-sequences represent partial solutions.

▶ **Definition 13.** *A family $R \subseteq \mathscr{P}$* represents *another family S, if it holds for all $q \in \mathscr{P}$ that q is consistent with a pattern of S iff if it is consistent with a pattern of R. The family R* parity-represents *S over a family $C \subseteq \mathscr{P}$, if it holds for each $q \in C$ that q is consistent with an odd number of patterns of S iff it is consistent with an odd number of patterns of R.*

We show in the full version (and from [8]) that pattern operations and actions preserve both representation and parity-representation over $\mathscr{P}_C$, in the sense that if R (parity-) represents S, then $\mathrm{Op}(R)$ (parity-)represents $\mathrm{Op}(S)$. It was also shown in [8] that the four actions over a pattern p with $|\operatorname{inc}(p)| \leq 4$ represent p. Therefore, one can show by induction over $\mathcal{T}$ that the family of patterns corresponding to valid action-sequences at a node x represents the family of patterns corresponding to valid solution-sequences at x, and hence, represents all partial solutions at x as well. The following result follows:

▶ **Corollary 14.** *Let $b \in \overline{B}$. There exist $w \in \overline{W}$, $c \in \mathscr{C}$ and a valid action-sequence τ of cost b and weight w at r generating the pair $([0], c)$, if and only if there exists a connected odd cycle transversal of size b in G containing v_0.*

All that is left to show is that the algorithm correctly counts (modulo 2) the number of valid action-sequences of cost b and weight w at each node x that generate a pair (p, c), where $p \sim q$ for some complete pattern $q \in \mathscr{P}_C$. In order to show this, we prove that $\mathrm{PREP}(p)$ parity-represents p over $\mathscr{P}_C$ for any $p \in \mathscr{P}_C$. Based on this, together with the fact that all defined operations preserve parity-representation over $\mathscr{P}_C$, we prove the following lemma:

▶ **Lemma 15.** *For each node $x \in V(\mathcal{T})$, for all values b, w, and all $c \in \mathscr{C}$ and $q \in \mathscr{P}_C$, it holds that $\sum_{\substack{p \in \mathscr{P}_{CS} \\ p \sim q}} T\big[x, b, w\big]\big[(p, c)\big] \equiv_2 1$, if and only if there exists an odd number of valid action-sequences τ at x of cost b and weight w generating the pair (p, c) with $p \sim q$.*

The following corollary follows directly from Lemma 6:

▶ **Corollary 16.** *It holds for all b, w, and all $c \in \mathscr{C}$ that $T\big[r, b, w\big]\big[([0], c)\big] = 1$ iff there exists an odd number of valid action-sequences of cost b and weight w at r generating $([0], c)$.*

So far, we have shown that the algorithm correctly counts the number of action-sequences of a specific weight, generating a pair $([0], c)$ for any coloring c. We make use of the isolation lemma [35], to show that by a careful choice of the value W and the weight function ω, we get a unique minimum weight sequence with high probability if a solution exists.

▶ **Lemma 17.** *Assume that G contains a connected odd cycle transversal of size $\overline{b}$ containing v_0. Fix $W = 8 \cdot |\mathcal{V}^{CJ}|$, and choose $\omega(x,i)$ independently and uniformly at random in $[W]$. Let τ be a minimum weight valid action-sequence of cost $\overline{b}$ generating a pair $([0],c)$ for any coloring c. Then τ is unique with probability at least $1/2$.*

Proof sketch of Theorem 1. Fix W and ω as in Lemma 17. For each $v \in V$ fix $v_0 = v$, and compute tables $T[x,b,w]$ for all $x \in \mathcal{V}$, and all b, w. The algorithm accepts if there exist a choice of v_0, a coloring $c \in \mathscr{C}$ and a weight w such that $T[r,\overline{b},w]\big[([0],c)\big] = 1$, and rejects otherwise. The algorithms runs in time $\mathcal{O}^*(12^{\mathrm{cw}})$ by Corollary 9.

The correctness follows by the following observation: There exists a solution of cost b if and only if, for any vertex v_0 in this solution, there exists a valid action-sequence at r of cost b generating the pair $([0],c)$ for some $c \in \mathscr{C}$. If such a sequence exists, then by Lemma 17 a unique minimum weight sequence exists with probability at least $1/2$. Hence, it follows from Corollary 16 that if a solution exists, then there exist w and $c \in \mathscr{C}$ such that $T[x,b,w]\big[([0],c)\big] = 1$ with probability at least $1/2$. On the other hand, if no such sequence exists, then it follows by Corollary 16 that $T[x,b,w]\big[([0],c)\big] = 0$ for all values w, any coloring c and any choice of v_0. ◀

6 Lower bound

We prove Theorem 2 through a parameterized reduction from the d-SAT problem for each constant d, proving that an algorithm running in time $\mathcal{O}^*((12-\varepsilon)^{\mathrm{cw}})$ for some $\varepsilon \geq 0$ contradicts SETH. Let I be the given instance with clauses $C_1, \ldots, C_m$ over variables $v_1, \ldots, v_n$. We refer to the full version for details. We start by a formal statement of SETH.

▶ **Conjecture 18** ([27, 28])**.** *For any positive value δ there exists an integer d such that the d-SAT problem cannot be solved in time $\mathcal{O}^*((2-\delta)^n)$, where n is the number of variables.*

Construction of the graph. We build G by combining copies of constant components, called gadgets. We distinguish three kinds of gadgets: *path gadgets*, *decoding gadgets*, and *clause gadgets*. We distinguish a vertex r in G called the *root*, that is forced to every solution, and another vertex $r_{\boldsymbol{B}}$. The connectivity of a solution can be proven by showing that all vertices are connected to r by a path. The vertex $r_{\boldsymbol{B}}$ will not be contained in any optimal solution, and its color serves as reference for a witness of an optimal solution.

Let t_0 be a constant that we fix later, and $s = \lceil n/t_0 \rceil$. We partition the variables into s groups $V_1, \ldots, V_s$ each of size t_0 (except for the last one). Let $t = \lceil t_0 \cdot \log_{12}(2) \rceil$ and $n' = s \cdot t$. The graph G consists of n' path-like structures, called *path sequences*. Each path sequence is a long sequence of path gadgets. Consecutive path gadgets are connected by small bicliques, giving a path sequence its narrow path-like structure. Let $P_1, \ldots, P_{n'}$ be the path sequences of G. We group them into s groups of size t each, calling each of these groups a *path bundle*. For a path bundle $\mathscr{B}_i$ and $j \in [c]$, we call the set of the jth path gadgets on each path sequences in $\mathscr{B}_i$ a *(simple) bundle* B_i^j. Finally, we call the set consisting of the j-th path gadget of each path sequence a *column* $\mathrm{Col}_j = \{X_i^j : i \in [n']\}$. We attach a decoding gadget D_i^j to each bundle and a clause gadget Z_j to each column. See Figure 1 for illustration.

In each path gadget X, We distinguish 3 entry vertices and three exit vertices. We add a biclique between the exit vertices of each path gadget, and the entry vertices of the following path gadget. We attach a special gadget to each entry or exit vertex of a path gadget called a *simple path gadget*. We define the set of *simple states* $\mathscr{I} = \{\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{B}, \boldsymbol{W}\}$. A solution defines a simple state in each simple path gadget. The states defined by the simple path

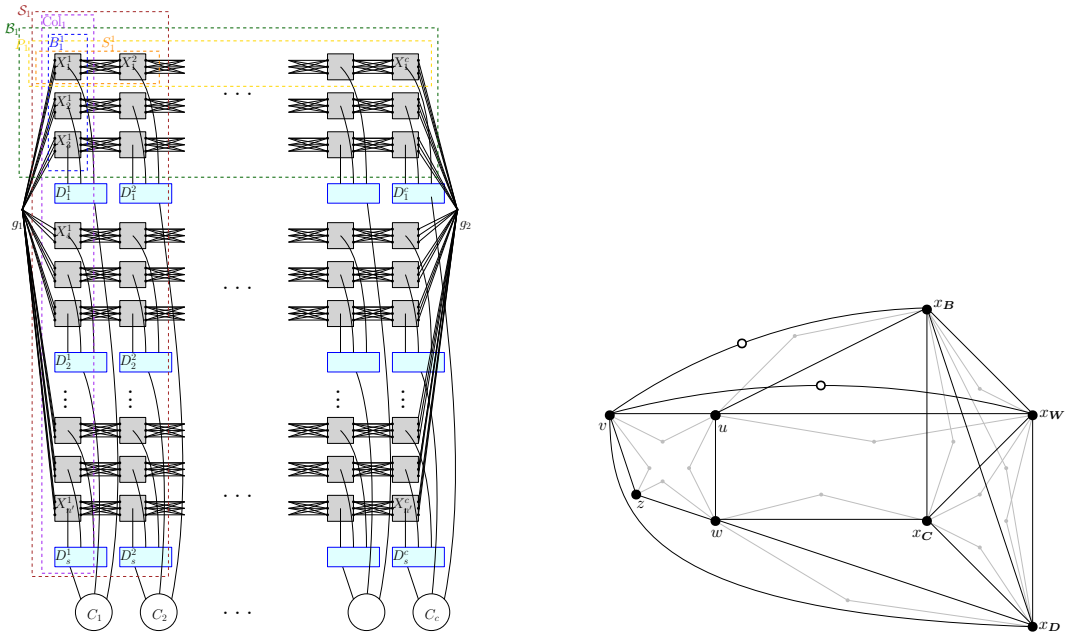

Figure 1 On the left, the graph G corresponding to an instance with two clauses. The gray squares represent path gadgets. Decoding gadgets are depicted in cyan, and clause gadgets as circles. We outline in yellow, blue, green, purple, orange and brown, the first path sequence, bundle, path bundle, column, segment and section respectively. On the right we depict a simple path gadget.

gadgets at the entry vertices of a path gadget combine to one of 12 specific *states* $\mathcal{S} \subseteq \mathscr{I}^3$ (see full version). The combination of states over a bundle B_i^j corresponds to a Boolean assignment over the variable group V_i. The clause gadget and all decoding gadgets on the jth column enforce a partial assignment of some variable group to satisfying a corresponding clause. Finally, the bicliques restrict transitions of states along consecutive path gadgets, ensuring that in any solution there exists a long segment where all path gadgets share the same state along each path. This ensures that the corresponding Boolean assignment satisfies all clauses.

Budget. We choose the value $\overline{b}$ in the output instance $(G, \overline{b})$ by defining a family $\mathcal{F}$ of pairwise vertex-disjoint components in G (mostly the gadgets of G). For each component $X \in \mathcal{F}$, we define a value $b(X)$. We define $\overline{b}$ as the sum of $b(X)$ for all $X \in \mathcal{F}$. In the full version, we show that any solution must contain $b(X)$ vertices in each component $X \in \mathcal{F}$. By the tightness of $\overline{b}$, it follows that a solution of size $\overline{b}$ contains exactly $b(X)$ vertices in each component $X \in \mathcal{F}$, and no other vertices.

Correctness. We show the correctness of our reduction in two steps. First we show that I has a satisfying assignment if and only if G admits a solution of size $\overline{b}$. Second, we show that the graph G has linear clique-width at most $k = n' + k_0$ for some constant k_0. Using these, we show that given an algorithm running in time $\mathcal{O}^*((12-\varepsilon)^k)$ for some $\varepsilon \geq 0$, we can choose the constant t_0 large enough, such that the d-SAT problem for any value of d can be solved in time $\mathcal{O}^*((2-\delta)^n)$ for some positive value δ, contradicting SETH.

7 Conclusion

In this work, we have provided a single-sided error Monte Carlo algorithm for the Connected Odd Cycle Transversal problem with running time $\mathcal{O}^*(12^{cw})$, presenting a new application of the "isolating a representative" technique by [6]. We proved that the running time is tight assuming SETH. This closes the second open problem posed by Hegerfeld and Kratsch [24]. However, one more interesting benchmark problem is still open, namely the Feedback Vertex Set problem. As mentioned in their paper, solving this problem using the Cut&Count technique is challenging, since a direct application would require counting edges induced by partial solutions. Another approach would be to follow the approach of Bojikian and Kratsch [8]. The difficulty of such application lays in finding a low GF(2)-rank representation of partial solutions.

Since both related techniques – the Cut&Count technique [15] and the rank-based approach [5] – were used to solve connectivity problems more efficiently under different structural parameters, it is natural to ask whether the approach of Bojikian and Kratsch [8] can likewise be adapted to different settings. In particular, one would ask whether this results in tight bounds for different connectivity problems under different parameters, where there also exists a gap between the rank of a corresponding compatibility matrix, and the size of a largest triangular submatrix thereof.

References

1 Isolde Adler and Mamadou Moustapha Kanté. Linear rank-width and linear clique-width of trees. *Theor. Comput. Sci.*, 589:87–98, 2015. doi:10.1016/j.tcs.2015.04.021.

2 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.

5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

6 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.

7 Narek Bojikian and Stefan Kratsch. Tight algorithm for connected odd cycle transversal parameterized by clique-width. *CoRR*, abs/2402.08046, 2024. doi:10.48550/arXiv.2402.08046.

8 Narek Bojikian and Stefan Kratsch. A tight monte-carlo algorithm for steiner tree parameterized by clique-width. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.29.

9 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

10 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

11 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

14 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

16 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

17 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

18 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

19 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

20 Frank Gurski and Egon Wanke. On the relationship between nlc-width and linear nlc-width. *Theor. Comput. Sci.*, 347(1-2):76–89, 2005. doi:10.1016/j.tcs.2005.05.018.

21 Frank Gurski and Egon Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.*, 359(1-3):188–199, 2006. doi:10.1016/j.tcs.2006.02.026.

22 Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.17.

23 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *CoRR*, abs/2302.03627, 2023. doi:10.48550/arXiv.2302.03627.

24 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 59:1–59:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.59.

25 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023. doi:10.1007/978-3-031-43380-1_28.

26 Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. *Discret. Appl. Math.*, 160(6):888–901, 2012. doi:10.1016/j.dam.2011.03.018.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

29 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

32 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

33 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *CoRR*, abs/1007.5450, 2010. arXiv:1007.5450.

34 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

35 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

36 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

37 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.