A Fast Parameterized Algorithm for Co-Path Set

Blair D. Sullivan, Andrew van der Poel
Department of Computer Science, North Carolina State University,
Raleigh, NC 27695
{blair.sullivan,ajvande4}@ncsu.edu
May 16, 2016

Abstract

The k-Co-Path Set problem asks, given a graph G and a positive integer k, whether one can delete k edges from G so that the remainder is a collection of disjoint paths. We give a linear-time fpt algorithm with complexity $O^*(1.588^k)$ for deciding k-Co-Path Set, significantly improving the previously best known $O^*(2.17^k)$ of Feng, Zhou, and Wang (2015). We also describe a new $O^*(4^{tw(G)})$ algorithm for Co-Path Set using the Cut&Count framework. In general graphs, we combine this with a branching algorithm which refines a 6k-kernel into reduced instances, which we prove have bounded treewidth.

1 Introduction

We study parameterized versions of Co-Path Set [3], an NP-complete problem asking for the minimum number of edges whose deletion from a graph results in a collection of disjoint paths (a co-path set). Specifically, we are concerned with k-Co-Path Set, which uses the natural parameter of the number of edges deleted.

$k$-Co-Path Set

Input: A graph $G = (V,E)$ and a non-negative integer $k$.
Parameter: $k$
Problem: Does there exist $F \subseteq E$ of size exactly $k$ such that $G[E \setminus F]$ is a set of disjoint paths?

These problems are naturally motivated by a special case of finding radiation hybrid mappings in genetics. These mappings are constructed to determine the ordering of genetic markers on DNA fragment data (created by breaking chromosomes with gamma radiation) [4,13,15]. Co-Path Set is equivalent to the setting where each fragment contains exactly two markers, giving a graph where edges in the co-path set obstruct a linear ordering.

Throughout this paper, we will use the notation $O^*(f(k))$ for $O(f(k)n^{O(1)})$ when denoting the complexity of fpt (fixed parameter tractable) algorithms; linear-fpt algorithms guarantee an upper bound of $O(f(k)n)$.

Recent algorithmic results include a $(10/7)$-approximation algorithm for Co-Path Set [2], and two randomized algorithms deciding k-Co-Path Set [6,7].
Figure 1: Three co-path sets (dashed edges), including one of minimum size (rightmost).

The faster parameterized algorithm $[7]$ has time complexity $O^*(2.17^k)$. However, as written, both parameterized results $[6,7]$ contain a flaw in their analysis which invalidates their probability of a correct solution in the given time $[1]$. The best known bound prior to $[6]$ is an $O^*(2.45^k)$ algorithm $[16]$. In this paper, we prove:

**Theorem 1.** $k$-Co-Path Set is decidable in $O^*(1.588^k)$ linear-FPT time.

After essential definitions and notation in Section 2, we start in Section 3 by giving a new $O^*(4^{tw(G)})$ algorithm $tw$-coPath for solving Co-Path Set parameterized by treewidth using the Cut&Count framework $[5]$. Finally, Section 4 describes a linear-FPT algorithm $copath$, which solves $k$-Co-Path Set on general graphs in $O^*(1.588^k)$ by applying $tw$-coPath to a set of “reduced instances” generated via kernelization and a branching procedure $deg$-branch.

2 Preliminaries

We write $G(V,E)$ for the graph with vertex set $V$ and edge set $E$. Unless otherwise noted, we assume $|V| = n$. The degree of a vertex $v$ is denoted $\text{deg}(v)$, and we let $\mathcal{N}(v)$ denote the set of neighbors. Given a graph $G(V,E)$ and $F \subseteq E$, we write $G[F]$ for the graph $G(V,F)$. In Co-Path Set problems, we want to find $F$ so that $G[E \setminus F]$ is a collection of disjoint paths (connected graphs with maximum degree 2 and $n - 1$ edges); we will refer to such $F$ as co-path sets (see Figure 1). We use $cc(G)$ to denote the number of connected components and $n_I(G)$ to denote the number of isolated vertices.

Our first algorithm for solving Co-Path Set uses dynamic programming over a tree decomposition, and its running time depends on the related measure of treewidth, which we denote $tw(G)$ (both introduced by Robertson & Seymour $[14]$ and heavily employed in parameterized complexity). To simplify the dynamic programming, we will use nice tree decompositions $[10]$, a more restrictive variant where each node of $T$ has one of five specific types: leaf, introduce vertex, introduce edge, forget vertex, or join. This formulation includes the perhaps unfamiliar “introduce edge” bags, which are labelled with an edge $uv$ and have one child (with an identical bag); we require each edge in $E$ is introduced exactly once. A tree decomposition can be transformed into a nice decomposition of the

---

1Step 2.11 in both versions of Algorithm R-MCP checks if a candidate co-path set $F$ has size $\leq k_1$ (as they are sweeping over all possible sizes of candidates and want to restrict the size accordingly). If $F$ is too large, the algorithm discards it and continues to the next iteration. However, in order for their analysis to hold, the probability that the candidate is contained in a co-path set must be $\geq \left(\frac{1}{2.17}\right)^{k_1}$ (or $\left(\frac{1}{2.29}\right)^{k_1}$ in $[6]$) for every iteration. Candidates which are too large may have significantly smaller probability of containment, yet are counted in the exponent of the analysis.

2The properties of our reduced instances guarantee we can find a tree decomposition in poly($k$) time.
We give an fpt algorithm for \( \text{Co-Path Set} \).

Theorem 2. There exists a one-sided fpt Monte Carlo algorithm with the number of consistent cuts, we introduce

4 Hamiltonian Cycle components and \( n \) has exactly 2 Hamiltonian Cycle of consistent cuts for subgraphs that are not collections of disjoint paths. We

saw that a graph \( G \) such cuts. Recall that \( \text{cc}(G) \) is the number of connected components and \( n_G(G) \) is the number of isolated vertices. In order to utilize parity with the number of consistent cuts, we introduce markers, which create even numbers of consistent cuts for subgraphs that are not collections of disjoint paths. We

The Cut&Count technique has two main ingredients: an algebraic approach to counting which uses arithmetic in \( \mathbb{Z}_2 \) alongside a guarantee that undesirable objects are seen an even number of times (so a non-zero result implies a desired solution has been seen), and the idea of defining the problem's connectivity requirement through consistent cuts. In this context, a consistent cut is a partitioning \( (V_1, V_2) \) of the vertices of a graph into two sets such that no edge \( uv \) has \( u \in V_1 \) and \( v \in V_2 \) and all vertices of degree 0 are in \( V_1 \). Since each connected component must lie completely on one side of any consistent cut, we see that a graph \( G \) has exactly \( 2^\text{cc}(G) - n_G(G) \) such cuts. We show:

Theorem 2. There exists a one-sided fpt Monte Carlo algorithm \( \text{tw-} \text{copath} \) deciding \( k\text{-Co-Path Set} \) for all \( k \) in a graph \( G \) in \( O^*(4^{\text{tw}(G)}) \) with failure probability \( \leq 1/2 \).

The Cut&Count technique has two main ingredients: an algebraic approach to counting which uses arithmetic in \( \mathbb{Z}_2 \) alongside a guarantee that undesirable objects are seen an even number of times (so a non-zero result implies a desired solution has been seen), and the idea of defining the problem's connectivity requirement through consistent cuts. In this context, a consistent cut is a partitioning \( (V_1, V_2) \) of the vertices of a graph into two sets such that no edge \( uv \) has \( u \in V_1 \) and \( v \in V_2 \) and all vertices of degree 0 are in \( V_1 \). Since each connected component must lie completely on one side of any consistent cut, we see that a graph \( G \) has exactly \( 2^\text{cc}(G) - n_G(G) \) such cuts. We show:

Theorem 2. There exists a one-sided fpt Monte Carlo algorithm \( \text{tw-} \text{copath} \) deciding \( k\text{-Co-Path Set} \) for all \( k \) in a graph \( G \) in \( O^*(4^{\text{tw}(G)}) \) with failure probability \( \leq 1/2 \).
further use weights and the Isolation Lemma to bound the probability of a false negative arising from multiple valid markings of a solution. To count the size of the collection of subgraphs with marked consistent cuts, tw-copath uses dynamic programming over a nice tree decomposition. We use fast subset convolution \cite{1} to reduce the complexity required for handling join bags. In the remainder of this section, we present the specifics for applying these techniques to solve Co-Path Set.

3.1 Cutting

We first provide formal definitions of markers and marked consistent cuts, which we use to ensure that sets of disjoint paths are counted exactly once in some entry of our dynamic programming table.

**Definition 2.** A triple \((V_1, V_2, M)\) is a marked consistent cut of a graph \(G\) if \((V_1, V_2)\) is a consistent cut and \(M \subseteq E(G[V_1])\). We refer to the edges in \(M\) as the markers. A marker set is proper if it contains at least one edge in each non-isolate connected component.

Note that proper marker sets determine unique marked consistent cuts, the latter are what we count in the dynamic programming algorithm described in the next section. We refer to the complement of a co-path set (the edges in the disjoint paths) as a cc-solution, and call it a marked-cc-solution when paired with a proper marker set of size exactly equal to its number of non-isolate connected components. While cc-solutions can be viewed as solutions due to their complementary nature, being marked is crucial in our counting algorithm and thus marked-cc-solutions are what we want to isolate.

We now describe our use of the Isolation Lemma, which guarantees we are able to use parity to distinguish solutions. Let \(f(X)\) denote \(\sum_{x \in X} f(x)\).

**Isolation Lemma** (\cite{11}). Let \(F \subseteq 2^U\) be a non-empty set family over universe \(U\). A function \(\omega: U \rightarrow \mathbb{Z}\) is said to isolate \(F\) if there is a unique \(S \in F\) with \(\omega(S) = \min_{F \in F} \omega(F)\). Assign weights \(\omega: U \rightarrow \{1, 2, \ldots, N\}\) uniformly at random. Then the probability that \(\omega\) isolates \(F\) is at least \(1 - |U|/N\).

Intuitively, if \(F\) is the set of solutions and \(|F|\) is even, then tw-copath would return a false negative. The Isolation Lemma allows us to partition \(F\) based on the weight of each solution, and guarantees at least one of the partition’s blocks has odd size with constant probability. Therefore, let \(F\) be the family of marked-cc-solutions and \(U\) contain two copies of every edge \(e \in E\): one representing \(e\) as a marker with weight \(\omega_M(e)\), and one as an edge in the cc-solution with weight \(\omega_E(e)\). Then \(2^U\) denotes all pairs of edge subsets (potential marked-cc-solutions), and we set \(N = 2|U| = 4E\). The probability of finding an isolating \(\omega\) is then \(1/2\), which allows us to guarantee success with high probability in a poly-logarithmic number of trials.

3.2 Counting

A marked-cc-solution corresponds to a co-path set of size \(k\) when its numbers of vertices, edges and markers match specific values which depend on \(k\). We now describe a dynamic programming (DP) algorithm over a nice tree decomposition which returns mod 2 the number of appropriately sized marked-cc-solutions in the root’s subtree (for a fixed \(k\)). Since no-instances have no appropriately
sized marked-cc-solutions, and yes-instances have at least one, odd parity for the number of marked-cc-solutions of size corresponding to $k$ implies a solution to the $k$-Co-Path Set instance must exist.

During the DP algorithm we actually count (for all values $(m, e)$) the number of cc-candidates, which are subgraphs $G' \subseteq G$ with the following properties:

1. $G'$ has maximum degree 2,
2. $G'$ has $e$ edges, and
3. $G'$ has a marked consistent cut with $m$ markers.

The following lemma justifies counting cc-candidates in place of marked-cc-solutions.

**Lemma 2.** The parity of the number of marked-cc-solutions in $G$ with $e$ edges and weight $w$ is the same as the parity of the number of cc-candidates $G' \subseteq G$ with $e$ edges, $|G'| - e - n_I(G')$ markers, and weight $w$.

**Proof.** Consider subgraph $G' \subseteq G$ with maximum degree 2 and $e$ edges. Let $M'$ be a marking of $G'$ such that $\omega_E(E(G')) + \omega_M(M') = w$. Assume first that $G'$ is a collection of paths. Note that $G'$ has $|G'| - e - n_I(G')$ connected components. If $M'$ is a proper marker set of $G'$, then $|M'| = |G'| - e - n_I(G')$ and $(G', M')$ has exactly one consistent cut. Therefore $(G', M')$ contributes one to both the number of marked-cc-solutions and cc-candidates. If otherwise $M'$ is not a proper marker set, then $(G', M')$ contains an unmarked connected component and has an even number of consistent cuts, and therefore contributes an even number to the count of cc-candidates and 0 to the number of marked-cc-solutions. And finally if $G'$ contains at least one cycle then $cc(G') > |G'| - e - n_I(G')$. Therefore at least one connected component does not contain a marker, and the number of consistent cuts is even, so the contribution to the count of cc-candidates is again even and the contribution to the count of marked-cc-solutions is 0. We conclude that the parity of the number of marked-cc-solutions and the parity of the number of cc-candidates is equal.

As usual we encode the option to include or exclude each edge as it is introduced during the bottom-up dynamic programming from the cc-candidates. Note that if a cc-candidate contains a cycle then the number of marked consistent cuts is even, and thus such candidates do not change the parity of the result.

| Variable | Parameter | Maximum value |
|----------|-----------|---------------|
| $a$      | # of non-isolated vertices | $n$           |
| $e$      | # of edges          | $n^2$         |
| $m$      | # of markers        | $n^2$         |
| $w$      | weight of edges and markers | $4n^4$       |

Table 1: Dynamic programming table parameters and upper bounds.

We distinguish between collections of cc-candidates using several parameters (see Table 1), as well as degree-functions, which keep track of the degrees of the vertices contained in the current bag. A degree-function maps the vertices of a bag to $\Sigma = \{0, 1, 1_2, 2\}$, according to their degree in a table entry’s corresponding
cc-candidates — for vertices of degree 1, their value $1_j$ denotes which side of the partition $(V_1, V_2)$ they are on. $A_x(a, e, m, w, s)$ is the entry in the table counting the number of cc-candidates at bag $x$ with a non-isolated vertices, $e$ edges, $m$ markers, weight $w$, and degree-function $s$.

We now give our dynamic programming algorithm for counting the number of cc-candidates in each table entry for all bags $x \in T$. In the following, we let $z_1, z_2$ denote the children of a join node; otherwise, the unique child is denoted $y$.

**Leaf:**

$$A_x(0, 0, 0, 0, \emptyset) = 1, \quad A_x(a, e, m, w, s) = 0$$

for all other inputs.

**Introduce vertex $v$:**

$$A_x(a, e, m, w, s) = [s(v) = 0]A_y(a, e, m, w, s).$$

**Introduce edge $uv$:**

$$A_x(a, e, m, w, s) = A_y(a, e, m, w, s) + \sum_{t \in \{u, v\}} \sum_{a \in \text{subs}(s(t))} \left( \phi_1(a, \alpha_v)A_y(a', e - 1, m, w', s') + \phi_2(a, \alpha_u)A_y(a', e - 1, m, w', s') \right),$$

where $\phi_j(a_u, \alpha_v) = (\alpha_u = 1_j \lor s(u) = 1_j) \land (\alpha_v = 1_j \lor s(v) = 1_j)$, $a' = a - ([11, 12] \cap \{s(u), s(v)\})$, $s' = s[u \to \alpha_u, v \to \alpha_v]$, and the $\text{subs}$ function returns all the states degree-function $s$ could yield in child $y$ (summarized below).

| $s(v)$ | $0$ | $1_1$ | $1_2$ | $2$ |
|--------|-----|-------|-------|----|
| $\text{subs}(s(v))$ | $\emptyset$ | $0$ | $0$ | $\{11, 12\}$ |

We are not forced to use edge $uv$ in a cc-solution, so we include $A_y(a, e, m, w, s)$ in our summation. We iterate over all possible $\text{subs}$ values for each endpoint, only considering counts in child $y$’s entries where $u$ and $v$ have the appropriate $\text{subs}$ values, preventing us from ever having a vertex with degree greater than 2. Note that we use the $\phi_j$ function to assure that if $u$ or $v$ is an isolate (as per $s$) we do not use the introduced edge. We have a summation for both possible $j$ values in order to consider $uv$ falling on either side of the cut. If we include $uv$ in a cc-solution, the formulation of $a'$ assures that each endpoint of degree 1 is now included in the count of non-isolates (i.e. when $u$ and/or $v$ had degree 0 in $y$). We utilize the marker weight of $uv$ to distinguish when we choose it as a marker (only if on $V_1$ side of cut), and increment $m$ accordingly. In either case, we update $w$ appropriately (with $w'$ if no marker, $w''$ if marker introduced).

**Forget vertex $h$:**

$$A_x(a, e, m, w, s) = \sum_{a \in \{0, 1, 1_2, 2\}} A_y(a, e, m, w, s[h \to a]).$$
As a forgotten vertex can have degree 0, 1 or 2 in a cc-candidate, we must consider all possible values that \( s \) assigns to \( h \) in child bag \( y \). Note that cc-candidates in which \( h \) is both not an isolate and not a member of a connected component that contains a marker will cancel, as \( h \) can be on either side of the cut and all parameters will be identical.

**Join:**

We compute \( A_2 \) from \( A_{z_1} \) and \( A_{z_2} \) with fast subset convolution, taking care to only combine table entries whose degree-functions are compatible.

**Definition 3.** At a join node \( x \) with children \( z_1 \) and \( z_2 \), the degree-functions \( s_1 \) from \( A_{z_1} \), \( s_2 \) from \( A_{z_2} \), and \( s \) from \( A_2 \) are compatible if one of the following holds for every vertex \( v \) in \( x \): (i) \( s_i(v) = 0 \) and \( s(v) = 2 \), \( i \neq l \) or (ii) \( s_1(v) = s_2(v) = 1 \) and \( s(v) = 2 \).

In order to apply Lemma \( \[ \] \) we let \( B \) be the bag at \( x \), and transform the values assigned by the degree function \( s \) to values in \( \mathbb{Z}_4 \). Let \( \phi: \{0,1,1,2\} \to \mathbb{Z}_4 \) and \( \rho: \{0,1,1,2\} \to \mathbb{Z} \) be defined as in the table below, extending to vectors by component-wise application.

|   | 0 | 1 | 1 | 2 |
|---|---|---|---|---|
| \( \phi \) | 0 | 1 | 3 | 2 |
| \( \rho \) | 0 | 1 | 1 | 2 |

We use this function \( \phi \) in order to apply Lemma \( \[ \] \) while the function \( \rho \) (which corresponds to a vertex’s degree) is used in tandem to ensure the compatibility requirements are met: if \( \phi(s_1) + \phi(s_2) = \phi(s) \), then necessarily \( \rho(s_1) + \rho(s_2) \geq \rho(s) \). From the above table it is easy to verify that \( \phi(s_1) + \phi(s_2) = \phi(s) \) and \( \rho(s_1) + \rho(s_2) = \rho(s) \) together imply that \( s_1, s_2 \) and \( s \) are compatible. We sum over both functions when computing values for join nodes, to make sure that solutions from the children are combined only when there is compatibility.

Assign \( t_1 = \phi(s_1) \), \( t_2 = \phi(s_2) \), and \( t = \phi(s) \) in accordance with Lemma \( \[ \] \). Let \( \rho(s) = \sum_{v \in B} \rho(s(v)) \); that is \( \rho(s) \) is the sum of the degrees of all the vertices in the join node, as assigned by degree-function \( s \). By defining functions \( f \) and \( g \) as follows:

\[ f^{(d,a,c,m,w)}(\phi(s)) = [\rho(s) = d]A_{z_1}(a,c,m,w,s), \]
\[ g^{(d,a,c,m,w)}(\phi(s)) = [\rho(s) = d]A_{z_2}(a,c,m,w,s), \]

and writing \( r_i^u \) for the vector \( \langle d_i, a_i, c_i, m_i, w_i \rangle \), we can now compute

\[ A_2(a,c,m,w,s) = \sum_{r_1^u + r_2^u = (\rho(s), a', c, m, w)} (f^{r_1^u} \ast g^{r_2^u})(\phi(s)) \]

where \( a' = a + |s_1^{-1}\{1,1\} \cap s_2^{-1}\{1,2\}| \). We point out that

\[ \sum_{r_1^u + r_2^u = (\rho(s), a', c, m, w)} (f^{r_1^u} \ast g^{r_2^u})(\phi(s)) = 1 \]

only if both \( \phi(s_1) + \phi(s_2) = \phi(s) \) and \( \rho(s_1) + \rho(s_2) = \rho(s) \); that is, exactly when \( s_1, s_2 \) and \( s \) are compatible.
We conclude this section by describing how we search the DP table for marked-cc-solutions at the root node \( r \). By Lemma 2, the parity of the number of marked-cc-solutions with \(|E| - k \) edges and weight \( w \) is the same as parity of the number of cc-candidates \( G' \) with \(|E| - k \) edges, \(|G'| - (|E| - k) - n_I(G')\) markers and weight \( w \). These candidates are recorded in the table entries \( A_r(a, |E| - k, a - |E| + k, w, \emptyset) \), where \( a \) is the number of non-isolates. Therefore, if there exists some \( a \) and \( w \) so that \( A_r(a, |E| - k, a - |E| + k, w, \emptyset) = 1 \), then we have a yes-instance of \( k\)-Co-Path Set.

By Lemma 1, the time complexity of \( \text{tw-copath} \) for a join node \( B \) is \( O^*(4^{|B|}) \), which is \( O^*(4^{tw}) \). Note that for the other four types of bags, we only consider one instance of \( s \) per table entry and thus the complexity for each is \( O^*(4^{tw}) \).

Since the nice tree decomposition has size linear in \( n \), the bottom-up dynamic programming runs in total time \( O^*(4^{tw}) \). This complexity bound combined with the correctness of \( \text{tw-copath} \) discussed above proves Theorem 2.

4 Achieving \( O^*(1.588^k) \) in General Graphs

In order to use \( \text{tw-copath} \) to solve \( k\)-CoPath Set in graphs with unbounded treewidth, we combine kernelization and a branching procedure to generate a set of reduced instances – bounded treewidth subgraphs of the input graph \( G \). Specifically, we begin by constructing a kernel of size at most \( 6k \) as described in [7]. Our reduced instances are bounded degree subgraphs of the kernel given by a branching technique. We prove that (1) at least one reduced instance is an equivalent instance; (2) we can bound the number of reduced instances; and (3) each reduced instance has bounded treewidth. Finally, we analyze the overall computational complexity of this process.

4.1 Kernelization and Branching

We start by describing our branching procedure \( \text{deg-branch} \) (Algorithm 1), which uses a degree-bounding technique similar to that of Zhang et al. [16]. Our implementation takes an instance \((G, k)\) of Co-Path Set and two non-negative integers \( \ell \) and \( D \), and returns a set of reduced instances \( \{(G_i, k - \ell)\} \) so that (1) each \( G_i \) is a subgraph of \( G \) with exactly \(|E| - \ell \) edges and maximum degree at most \( D \), and (2) at least one \((G_i, k - \ell)\) is an equivalent instance to \((G, k)\). The size of the output (and hence the running time) of \( \text{deg-branch} \) depends on both input parameters \( \ell \) and \( D \). We will select \( D \) to achieve the desired complexity in \( \text{copath} \) in Section 4.3.

Our branching procedure leverages the observation that if a co-path set \( S \) exists, then every vertex has at most two incident edges not in \( S \). Specifically, for every vertex of degree greater than \( D \), we branch on pairs of incident edges which could remain after removing a valid co-path set (calling each pair a candidate), creating a search tree of subgraphs.

Algorithm 1 returns the set of reduced instances, the size of which is at most the number of leaves in the search tree of the branching process (inequality can result from the algorithm discarding branches in which the number of edits necessary to branch on a vertex exceeds the number of allowed deletions remaining). The following Lemma gives an upper bound on the size of this set.
Algorithm 1: Generating reduced instances

1. Algorithm deg-branch($G, \ell, D, k$)
2. Let $v$ be a vertex of maximum degree in $G$
3. if $\text{deg}(v) \geq D + 1$ and $\ell \geq D - 1$ then
4. Select vertices $u_1, \ldots, u_{D+1}$ uniformly at random from $N(v)$
5. $R = \emptyset$, $E_v = \{\{v, u_i\}\}$
6. for $e_1, e_2 \in E_v, e_1 \neq e_2$ do
7. $E'_v = E_v \{e_1, e_2\}$
8. $R = R \cup \text{deg-branch}(G \setminus E'_v, \ell - (D - 1), D, k)$
9. return $R$
10. else if $\ell = 0$ and $\text{deg}(v) \leq D$ then return $(G, k - \ell)$
11. else return $\emptyset$ // Discard $G$

Lemma 3. Let $T$ be a search tree formed by deg-branch($G, \ell, D, k$). The number of leaves of $T$ is at most $\left(\frac{D+1}{2}\right)^{\ell/(D-1)}$.

Proof of Lemma 3. The number of children of each interior node of $T$ is $\left(\frac{D+1}{2}\right)^{\text{depth}(T)}$, resulting in at most $\left(\frac{D+1}{2}\right)^{\text{depth}(T)}$ leaves. The depth of $T$ is limited by the second condition of the if on line 3 of Algorithm 1. For each recursive call, $\ell$ is decremented by $(D - 1)$, until $\ell \leq D - 1$. This implies $\text{depth}(T) \leq \ell/(D-1)$, proving the claim.

4.2 Treewidth of Reduced Instances

Our algorithm deg-branch produces reduced instances with bounded degree; in order to bound their treewidth, we make use of the following result, which originated from Lemma 1 in [8] and was extended in [9].

Lemma 4. For $\epsilon > 0$, there exists $n_\epsilon \in \mathbb{Z}^+$ s.t. for every graph $G$ with $n > n_\epsilon$ vertices,

$$\text{tw}(G) \leq \left(\sum_{i=3}^{17} c_i n_i\right) + n_{\geq 18} + \epsilon n,$$

where $n_i$ is the number of vertices of degree $i$ in $G$ for $i \in \{3, \ldots, 17\}$, $n_{\geq 18}$ is the number of vertices of degree at least 18, and $c_i$ is given in Table 2. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time.

Since the structure of $k$-Co-Path Set naturally provides some constraints on the degree sequence of yes-instances, we are able to apply this to our reduced instances to effectively bound treewidth. We first find an upper bound on the number of degree-3 vertices in any yes-instance of $k$-Co-Path Set.
Table 2: Numerically obtained constants $c_d$, $3 \leq d \leq 17$, used in Lemma 4 from Table 6.1 in [9].

| $d$ | $3$ | $4$ | $5$ | $6$ | $7$ | $8$ | $9$ | $10$ |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $c_d$ | 0.1667 | 0.3334 | 0.4334 | 0.5112 | 0.5699 | 0.6163 | 0.6538 | 0.6847 |

| $d$ | $11$ | $12$ | $13$ | $14$ | $15$ | $16$ | $17$ |
|-----|-----|-----|-----|-----|-----|-----|-----|
| $c_d$ | 0.7105 | 0.7325 | 0.7514 | 0.7678 | 0.7822 | 0.7949 | 0.8062 |

Lemma 5. Let $n_i$ be the number of vertices of degree $i$ in a graph $G$ for any $i \in \mathbb{Z}^+$, and $\Delta$ be the maximum degree of $G$. If $(G, k)$ is a yes-instance of $k$-Co-Path Set, then $n_3 \leq 2k - (\sum_{i=4}^{\Delta} (i - 2)n_i)$.

Proof. Since $(G, k)$ is a yes-instance, removing some set of at most $k$ edges results in a graph of maximum degree 2. For a vertex of degree $j \geq 3$, at least $j - 2$ incident edges must be removed. Thus, $n_3 + 2n_4 + 3n_5 + \ldots + (\Delta - 2)n_\Delta \leq 2k$ (each removed edge counts twice – once for each endpoint).

Corollary 1. Let $(G, k)$ be an instance of $k$-Co-Path Set such that $G$ has $n$ vertices and max degree at most $\Delta \in \{3, \ldots, 17\}$. Then the treewidth of $G$ is bounded by $k/3 + \epsilon n + c$, for some constant $c$ and any $\epsilon > 0$.

Proof. Let $G'$ be the graph formed by adding $N$ isolates to $G$. By Lemma 4 because $G'$ has maximum degree at most $\Delta$, $tw(G') \leq (1/6)n_3 + (1/3)n_4 + \ldots + c_\Delta n_\Delta + \epsilon(N + n)$. We can substitute the bound for $n_3$ from Lemma 5 which yields:

$$tw(G') \leq \frac{2k - (\sum_{i=4}^{\Delta} (i - 2)n_i)}{6} + \frac{n_4}{3} + \ldots + c_\Delta n_\Delta + \epsilon(N + n) \leq \frac{k}{3} + \epsilon(n + N).$$

Note that the inequality holds because we can pair the negative terms of $(\sum_{i=4}^{\Delta} (i - 2)n_i)/6$ with the corresponding terms of $n_4/3 + \ldots + c_\Delta n_\Delta$ and the value of $c_j n_j - (j - 2)(n_j)/6$ is non-positive for all $j \in [4, 17]$. Since $N = n_\epsilon$ is a constant, we have $tw(G') \leq k/3 + \epsilon n + c$. Since $G \subseteq G'$ and treewidth is monotone under subgraph inclusion, this proves the claim.

We point out that when applying Corollary 1 to reduced instances, computing a tree decomposition is polynomial in $k$ (since they are subgraphs of a $6k$-kernel).

### 4.3 The Algorithm copath

This section describes how we combine the above techniques to prove Theorem 1. As shown in Algorithm 2, we start by applying 6k-kernel [7] to find $G'$, a kernel of size at most 6k; this process deletes $k - k'$ edges. We then guess the number of edges $k_1 \in [0, k']$ to remove during branching, and use deg-branch to create a set of reduced instances $Q_{k_1}$, each of which have $k' - k_1$ edges. To ensure the complexity of finding the reduced instances does not dominate the running time, we set the degree bound $D$ of the reduced instances to be 10 (any choice of $10 \leq D \leq 17$ is valid). By considering all possible values of $k_1$, we are assured
that if \((G', k')\) is a yes-instance, some \(Q_{k_1}\) contains a yes-instance. Each reduced instance is then passed to \textit{tw-copath}, which correctly decides the problem with constant probability.

Algorithm 2: Deciding \(k\)-Co-Path Set

\begin{algorithmic}
1 Algorithm copath \((G, k)\)
2 \((G', k') = \text{6k-kernel}(G, k)\)
3 for \(k_1 \leftarrow 0\) to \(k'\) do
4 \(Q_{k_1} = \text{deg-branch}(G', k_1, 10, k)\)
5 foreach \((G_i, k_2) \in Q_{k_1}\) do
6 \textbf{if} \textit{tw-copath}(\(G_i, k_2\)) \textbf{then} return true
7 return false
\end{algorithmic}

Proof of Theorem 1. We now analyze the running time of copath, as given in Algorithm 2. By Lemma 3, the size of each \(Q_{k_1}\) is \(O(1.561^k)\). For each reduced instance in \((G_i, k_2)\) in \(Q_{k_1}\), we have \(\text{tw}(G_i) \leq k_2/3 + \epsilon(6k) + c\) by Corollary 1.

Applying Theorem 2, \textit{tw-copath} runs in time \(O^*(4^{k_2/3+\epsilon6k})\) for each reduced instance \((G_i, k_2)\) in \(Q_{k_1}\). Each iteration of the outer \textbf{for} loop can then be completed in time

\[O^*(1.561^k4^{k_2/3+\epsilon6k}) = O^*(4^{k/3+\epsilon6k}) = O^*(1.588^k),\]

where we use that \(k_1 + k_2 = k' \leq k\), and choose \(\epsilon < 10^{-5}\). Since this loop runs at most \(k + 1\) times, this is also a bound on the overall computational complexity of \textit{copath}. Additionally \textit{copath} is linear-fpt, as the kernelization of [7] is \(O(n)\), and the kernel has size \(O(k)\), avoiding any additional \(\text{poly}(n)\) complexity from the \textit{tw-copath} subroutine.

5 Conclusion

This paper gives an \(O^*(4^w)\) fpt algorithm for Co-Path Set. By coupling this with kernelization and branching, we derive an \(O^*(1.588^k)\) linear-fpt algorithm for deciding \(k\)-Co-Path, significantly improving the previous best-known result of \(O^*(2.17^k)\). We believe that the idea of combining a branching algorithm which guarantees equivalent instances with bounds on the degree sequence from the problem’s constraints can be applied to other problems in order to obtain a bound on the treewidth (allowing treewidth-parameterized approaches to be extended to general graphs).

One natural question is whether similar techniques extend to the generalization of Co-Path Set to \(k\)-uniform hypergraphs (as treated in Zhang et al. [16]). It is also open whether the dual parameterization asking for a co-path set of size \(k\) resulting in \(\ell\) disjoint paths is solvable in sub-exponential fpt time.

Acknowledgements

This work supported in part by the Gordon & Betty Moore Foundation under DDD Investigator Award GBMF4560 and the DARPA GRAPHS program under SPAWAR Grant N66001-14-1-4063. Any opinions, findings, and conclusions or recommendations expressed in this publication
are those of the author(s) and do not necessarily reflect the views of DARPA, SSC Pacific, or the Moore Foundation. We thank two anonymous reviewers for providing a simplification of our previous branching algorithm and pointing out the result from [9] enabling us to branch on vertices with degree greater than 7. We also thank Felix Reidl for helpful suggestions on an earlier draft that significantly improved the presentation of the results.

References

[1] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In Proceedings of STOC, pages 67–74, 2007.

[2] Z. Chen, G. Lin, and L. Wang. An approximation algorithm for the minimum co-path set problem. Algorithmica, 60(4):969–986, 2011.

[3] Y. Cheng, Z. Cai, R. Goebel, G. Lin, and B. Zhu. The radiation hybrid map construction problem: recognition, hardness, and approximation algorithms. Unpublished Manuscript, 2008.

[4] D. Cox, M. Burmeister, E. Price, S. Kim, and R. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. Science, 250:245–50, 1990.

[5] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. van Rooij, and J. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In FOCS, pages 150–159. IEEE, 2011.

[6] Q. Feng, Q. Zhou, and S. Li. Randomized parameterized algorithms for co-path set problem. In FAW, pages 82–93. Springer, 2014.

[7] Q. Feng, Q. Zhou, and J. Wang. Kernelization and randomized parameterized algorithms for co-path set problem. J. Comb. Optim., pages 1–12, 2015.

[8] F. Fomin, S. Gaspers, S. Saurabh, and A. Stepanov. On two techniques of combining branching and treewidth. Algorithmica, 54(2):181–207, April 2009.

[9] S. Gaspers. Exponential Time Algorithms - Structures, Measures, and Bounds. VDM, 2010.

[10] T. Kloks. Treewidth, Computations and Approximations, volume 842 of LNCS. Springer, 1994.

[11] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. In Proceedings of STOC, pages 345–354. ACM, 1987.

[12] M. Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: a logical approach. CoRR, abs/1104.3057, 2011.

[13] C. Richard III, D. Withers, T. Meeker, S. Maurer, G. Evans, R. Myers, and D. Cox. A radiation hybrid map of the proximal long arm of human chromosome 11 containing the multiple endocrine neoplasia type 1 (men-1) and bcl-1 disease loci. Am. J. Hum. Genet., 49(6):1189–1196, 1991.

[14] N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms, 7(3):309–322, 1986.

[15] D. Slonim, L. Kruglyak, L. Stein, and E. Lander. Building human genome maps with radiation hybrids. J. Comp. Biol., 4(4):487–504, 1997.

[16] C. Zhang, H. Jiang, and B. Zhu. Radiation hybrid map construction problem parameterized. In Combin. Opt. and App., volume 7402 of LNCS, pages 127–137. Springer, 2012.