A Fast Parameterized Algorithm for Co-Path Set*

Blair D. Sullivan[1] **and Andrew van der Poel**[2]

1 **Dept. of Computer Science, North Carolina State University, Raleigh, USA**
`blair_sullivan@ncsu.edu`
2 **Dept. of Computer Science, North Carolina State University, Raleigh, USA**
`ajvande4@ncsu.edu`

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, randomized 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). Our main tool is a new $O^*(4^{tw(G)})$ algorithm for Co-Path Set using the Cut&Count framework, where $tw(G)$ denotes treewidth. In general graphs, we combine this with a branching algorithm which refines a $6k$-kernel into reduced instances, which we prove have bounded treewidth.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases co-path set, parameterized complexity, treewidth, fixed-parameter tractable complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.28

1 Introduction

We study parameterized versions of Co-Path Set [3, 16], an NP-complete problem asking for the minimum number of edges whose deletion from a graph results in a collection of disjoint paths (the deleted edges being a *co-path set* – see Figure 1). 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 determining the ordering of genetic markers in DNA using fragment data created by breaking chromosomes with gamma radiation (a technique known as *radiation hybrid mapping*) [4, 13, 15]. Unfortunately, human error in distinguishing markers often means the constraints implied by markers' co-occurrence on fragments are incompatible with all possible linear orderings, necessitating an algorithm to find the "best" ordering (that violates the fewest constraints). Co-Path Set solves the

---

* 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.

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin; Article No. 28; pp. 28:1–28:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

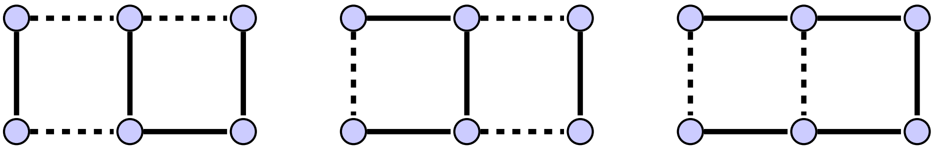

Figure 1 Three co-path sets (dashed edges), including one of minimum size (rightmost).

special case where each DNA fragment contains exactly two genetic markers (corresponding to an edge in the graph); any linear ordering of the markers must correspond to some set of paths, and we minimize the number of unsatisfied constraints (edges in the co-path set).

Recent algorithmic results related to Co-Path Set include a (10/7)-approximation algorithm [2], and two parameterized algorithms deciding k-Co-Path Set [6, 7], the faster of which [7] has time complexity[1] $O^*(2.17^k)$. 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[2]. 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 with probability at least $2/3$.*

We note that standard amplification arguments apply, and Theorem 1 holds for any success probability less than 1. Further, if f is an increasing function with $\lim_{n\to\infty} f(n) = 1$, we can solve k-Co-Path Set with success probability at least $f(n)$ in $O(1.588^k n \mathsf{polylog}(n))$.

The remainder of this paper is organized as follows: 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 (tw) using the Cut&Count framework [5]. Finally, Section 4 describes the linear-fpt algorithm referenced in Theorem 1, 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[3] `deg-branch`.

2 Preliminaries

Let $G(V, E)$ be the graph with vertex set V and edge set E. Unless otherwise noted, we assume $|V| = n$ and $|E| = m$; we let $N(v)$ denote the set of neighbors of a vertex v, and let $\deg(v) = |N(v)|$. Given a graph $G(V, E)$ and $F \subseteq E$, we write $G[F]$ for the graph $G(V, F)$.

Our `tw-copath` algorithm in Section 3 uses dynamic programming over a *tree decomposition*, and its running time depends on the related measure of *treewidth* [14], which we denote $tw(G)$. To simplify the dynamic programming, we will use a variant of *nice tree decompositions* [10, 5] where each node in the tree has one of five specific types: leaf, introduce

---

[1] Throughout this paper, we use the notation $O^*(f(k))$ for the fpt (fixed-parameter tractable) complexity $O(f(k)n^{O(1)})$; we say an algorithm is *linear-fpt* if the complexity is $O(f(k)n)$.

[2] Step 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 $\geq (1/2.17)^{k_1}$ (or $(1/2.29)^{k_1}$ 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.

[3] The properties of our reduced instances guarantee we can find a tree decomposition of small width in poly(k) time.

vertex, introduce edge, forget vertex, or join. The "introduce edge" nodes are labelled with an edge uv and have one child (with an identical bag); we require that each edge in E is introduced exactly once. Additionally, we enforce that the root node is of type "forget vertex" (and thus has an empty bag). A tree decomposition can be transformed into a nice decomposition of the same width in time linear in the size of the input graph [5].

When describing the dynamic programming portion of the algorithm we use Iverson's bracket notation: if p is a predicate we let $[\![p]\!]$ be 1 if p is true and 0 otherwise. We also use the shorthand $f[x \to y]$ to denote updating a function f so that $f(x) = y$ and all other values are unchanged.

Finally, we use fast subset convolution [1] to reduce the complexity of handling join nodes in the nice tree decomposition (Section 3). This technique maps functions of the vertices in a join bag to values in $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ (where p is chosen based on the application). The key complexity result we rely on uses the $\mathbb{Z}_p$ product, which is defined below. We write $\mathbb{Z}_p^B$ for the set of all vectors t of length $|B|$ assigning a value $t(b) \in \mathbb{Z}_p$ to each element of $b \in B$.

▶ **Definition 2** ($\mathbb{Z}_p$ **product).** Let $p \geq 2$ be a fixed integer and let B be a finite set. For $t_1, t_2, t \in \mathbb{Z}_p^B$ we say that $t_1 + t_2 = t$ if $t_1(b) + t_2(b) = t(b)$ (in $\mathbb{Z}_p$) for all $b \in B$. For a ring R and functions $f, g : \mathbb{Z}_p^B \to R$, define the $\mathbb{Z}_p$ product, $*_x^p$ as

$$(f *_x^p g)(t) = \sum_{t_1+t_2=t} f(t_1)g(t_2)\,.$$

Fast subset convolution guarantees that certain $\mathbb{Z}_p$ products can be computed quickly.

▶ **Lemma 3** (Cygan et al. [5]). *Let $R = \mathbb{Z}$ or $R = \mathbb{Z}_q$ for some constant q. The $\mathbb{Z}_4$ product of functions $f, g : \mathbb{Z}_4^B \to R$ can be computed in $4^{|B|}|B|^{O(1)}$ time and ring operations.*

3 An $O^*(4^{tw(G)})$ Algorithm via Cut&Count

We start by giving an fpt algorithm for Co-Path Set parameterized by treewidth. Our primary tool is the Cut&Count framework, which enables $c^{tw}n^{O(1)}$ one-sided Monte Carlo algorithms for connectivity-type problems with constant probability of a false negative. Cut&Count has previously been used to improve the best-known bounds for several well-studied problems, including Connected Vertex Cover, Hamiltonian Cycle, and Feedback Vertex Set [5]. Pilipczuk showed that an $O^*(c^{tw})$ algorithm for some constant c can be designed with the Cut&Count approach for Co-Path Set because the problem can be expressed in the specialized graph logic known as ECML+C [12]. However, since our end goal is to improve on existing algorithms for k-Co-Path Set in general graphs using a bounded treewidth kernel, we need to develop a specialized dynamic programming algorithm with a small value of c. We show:

▶ **Theorem 4.** *There exists a one-sided fpt Monte Carlo algorithm* `tw-copath` *deciding k-Co-Path Set for all k in a graph G in $O^*(4^{tw(G)})$ time with failure probability $\leq 1/3$, when a tree decomposition of width tw is given as input.*

The Cut&Count technique has two main ingredients: an algebraic approach to counting which uses arithmetic in $\mathbb{Z}_2$ (enabling faster algorithms) 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 $2^{cc(G)-n_I(G)}$ such cuts, where $cc(G)$ is the number of connected components and $n_I(G)$ is the number of isolates (vertices with degree 0). In order to utilize parity with the number of consistent cuts, we introduce *markers*, which create even numbers of consistent cuts for graphs that are not collections of disjoint paths. Our counting algorithm `tw-copath`, which computes the parity of the size of the collection of subgraphs with consistent cuts which adhere to specific properties pertaining to CO-PATH SET, employs dynamic programming over a nice tree decomposition. We further use weights and the Isolation Lemma to bound the probability of a false negative arising from multiple valid markings of a solution. We use fast subset convolution [1] to reduce the complexity required for handling join bags in the dynamic programming. 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 during our dynamic programming.

▶ **Definition 5.** 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 connected component of G which is not an isolate.

Note that if a marked consistent cut contains a proper marker set, all vertices are on the V_1 side of the cut. This is because by the definition of a consistent cut, all isolates are on the V_1 side, and if every connected component contains a marker then all connected components must fall entirely on the V_1 side as well. Therefore for any proper marker set there exists exactly one consistent cut, while all marker sets which are not proper will be paired with an even number of consistent cuts because unmarked components may lie in V_1 *or* V_2. We use proper marker sets to distinguish desired subgraphs by assigning markers in such a way that when we prune the dynamic programming table for solutions (as described later in the section), the only subgraphs we consider which may have a proper marker set are collections of disjoint paths. We know because the marker set is proper that the subgraph has a unique consistent cut, and thus these collections of disjoint paths will only be counted once in some entry of the dynamic programming table, while all other subgraphs will be counted an even number of times. Note that we are not claiming that all collections of disjoint paths will have proper marker sets.

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 subgraphs which are marked-cc-solutions are what correspond to solutions in the dynamic programming table.

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** ([11]). *Let $\mathcal{F} \subseteq 2^U$ be a non-empty set family over universe U. A function $\omega \colon U \to \mathbb{Z}$ is said to* isolate *$\mathcal{F}$ if there is a unique $S \in \mathcal{F}$ with $\omega(S) = \min_{F \in \mathcal{F}} \omega(F)$. Assign weights $\omega \colon U \to \{1, 2, ..., N\}$ uniformly at random, where the value of N is of the reader's choice. Then the probability that ω isolates $\mathcal{F}$ is at least $1 - |U|/N$.*

Intuitively, if $\mathcal{F}$ is the set of solutions (or complements of solutions) to an instance of Co-Path Set and $|\mathcal{F}|$ is even, then `tw-copath` would return a false negative. This is because while each solution is counted an odd number of times in `tw-copath`, because there are an even number of solutions the total count of solutions is even, making the combined count of solutions and non-solutions even and the algorithm would incorrectly determine a solution does not exist (a false negative). The Isolation Lemma allows us to partition $\mathcal{F}$ based on the weight of each solution (as assigned by ω), and guarantees at least one of the partition's blocks has odd size with constant probability. We let U contain two copies of every edge $e \in E$: one representing e as a marker and one as an edge in the cc-solution. Then 2^U denotes all pairs of edge subsets (potential marked-cc-solutions), and we set $N = 3|U| = 6E$ (selected to achieve success probability in Theorem 1). Each copy of an edge is assigned a weight in $[1, N]$ uniformly at random by ω and the probability of finding an isolating ω is thus $2/3$. We denote the values assigned by ω to the set of marker copies by ω_M, and likewise to the set of edge-in-cc-solution copies by ω_E.

3.2 Counting

A marked-cc-solution C of a graph G corresponds to a co-path set of size k when the number of edges and markers in C match specific values which depend on k and $|E(G)|$. These values are easily deduced because we know the deletion of a co-path set solution of size k will leave $|E(G)| - k$ edges in a cc-solution. Furthermore, because a forest has $n - m$ connected components, the number of markers in C needs to be at most $|V(C)| - |E(G)| + k$. All isolates from a forest can be removed and the resulting graph is still a forest, and thus the actual number of markers necessary in C is $|V(C)| - n_I(C) - |E(G)| + k$.

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 maximum degree 2, exactly e edges, and a marked consistent cut with m markers. The following lemma justifies counting cc-candidates in place of marked-cc-solutions. Note that the weight of a marked-cc-solution or a cc-candidate is equal to the sum of its marker weights and its edge weights.

▶ **Lemma 6.** *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, $|V(G')| - e - n_I(G')$ markers, and weight w.*

Proof. Consider a 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. We know that G' has $|V(G')| - e - n_I(G')$ non-isolate connected components. If M' is a proper marker set of G', then $|M'| = |V(G')| - e - n_I(G')$ and (G', M') has exactly one consistent cut. Therefore (G', M') contributes one to both the number of marked-cc-solutions and the number of cc-candidates, respectively.

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 zero to the number of marked-cc-solutions. Finally, if G' contains at least one cycle then $cc(G') > |V(G')| - e - n_I(G')$. 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

Table 1 Dynamic programming table parameters and upper bounds.

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	$4n^4$

to the count of marked-cc-solutions is zero. We conclude that the parity of the number of marked-cc-solutions and the parity of the number of cc-candidates is the same. ◀

Our dynamic programming algorithm is a bottom-up approach over a nice tree decomposition. We build cc-candidates for all values of m and e (encoding the option to add/not add edges and select/not select edges as markers), and keep track of various parameters ensuring that when pruning the DP table we only consider cc-candidates which could be valid solutions to the k-Co-Path Set instance. We use the number of edges to ensure our solution is of the correct size, and the number of markers and non-isolate vertices to determine when a subgraph is acyclic. The weight parameter allows us to distinguish between solutions and decreases the likelihood of a false negative occurring via the Isolation Lemma.

Finally, we need a parameter that encodes the degree information required to properly combine cc-candidates as we iterate up the tree. We call this parameter a *degree-function* and define it on the vertices V of a bag as $f : V \to \Sigma = \{0, 1_1, 1_2, 2\}$, where $f(v)$ corresponds to v's degree in the associated cc-candidates of the table entry — for vertices of degree 1, their value 1_j denotes which side of the partition (V_1, V_2) they are on. Vertices with degree 0 are on the V_1 side of the cut by definition and degree 2 vertices cannot gain additional incident edges, so we need not keep track of their side of the cut. In summary, we have table entries $A_x(a, e, m, w, s)$ counting the number of cc-candidates with a non-isolated vertices, e edges, m markers, weight w, and degree-function s, where all vertices which have been introduced in the subtree rooted at x are present and only edges which have been introduced in this subtree may be present.

In the following description of the dynamic programming algorithm over a nice tree decomposition T, we let z_1, z_2 denote the children of a join node; otherwise, the unique child is denoted y.

Leaf:

$A_x(0, 0, 0, 0, \emptyset) = 1$; $A_x(a, e, m, w, s) = 0$ for all other inputs.

Introduce vertex v:

$$A_x(a, e, m, w, s[v \to 0]) = A_y(a, e, m, w, s);\ A_x(a, e, m, w, s[v \to i]) = 0,\ \forall i \neq 0.$$

Introduce edge uv:

$$A_x(a, e, m, w, s) = A_y(a, e, m, w, s) + \sum_{\substack{\alpha_t \in subs(s(t)) \\ t \in \{u,v\}}} [\![\phi_2(\alpha_u, \alpha_v)]\!] A_y(a', e-1, m, w', s')$$

$$+ \sum_{\substack{\alpha_t \in subs(s(t)) \\ t \in \{u,v\}}} [\![\phi_1(\alpha_u, \alpha_v)]\!] \Big(A_y(a', e-1, m, w', s') + A_y(a', e-1, m-1, w'', s') \Big),$$

where $\phi_j(\alpha_u, \alpha_v) = (\alpha_u = 1_j \vee s(u) = 1_j) \wedge (\alpha_v = 1_j \vee s(v) = 1_j)$, $a' = a - (|\{1_1, 1_2\} \cap \{s(u), s(v)\}|)$, $w' = w - \omega_E(uv)$, $w'' = w - \omega_E(uv) - \omega_M(uv)$, $s' = s[u \to \alpha_u, v \to \alpha_v]$, and the *subs* function returns all the values the degree-function in child node y could have assigned to vertices u and v based on current degree-function s (summarized below).

$s(v)$	0	1_1	1_2	2
$subs(s(v))$	$\emptyset$	0	0	$\{1_1, 1_2\}$

We now argue this formula's correctness. The term $A_y(a, e, m, w, s)$ handles the case when uv is excluded from the cc-solution. We handle the case when uv is added to the cc-solution by iterating over all possible *subs* values for each endpoint, only considering counts in child y's entries where u and v have the appropriate *subs* values (preventing us from ever having a vertex with degree greater than 2). Note that we use the ϕ_j function to guarantee that if s labels u or v as an isolate, 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. 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:

$$A_x(a, e, m, w, s) = \sum_{\alpha \in \{0, 1_1, 1_2, 2\}} A_y(a, e, m, w, s[h \to \alpha]).$$

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 mod 2, as h can be on either side of the cut and all parameters will be identical.

Join: We compute A_x from A_{z_1} and A_{z_2} via fast subset convolution [1] taking care to only combine table entries whose degree-functions are *compatible*, ensuring that only joins which preserve the constraints of the degree-functions of the children nodes occur.

▶ **Definition 7.** 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_x are **compatible** if one of the following holds for every vertex v in x: (i) $s_i(v) = 0$ and $s_l(v) = s(v), i \neq l$ or (ii) $s_1(v) = s_2(v) = 1_j$ and $s(v) = 2$ for $i, j, l \in [1, 2]$.

In order to apply Lemma 3, 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 $\phi\colon \{0, 1_1, 1_2, 2\} \to \mathbb{Z}_4$ and $\rho\colon \{0, 1_1, 1_2, 2\} \to \mathbb{Z}$ be defined as in the table below, extending to vectors by component-wise application.

	0	1_1	1_2	2
ϕ	0	1	3	2
ρ	0	1	1	2

We use ϕ to apply Lemma 3, while the function ρ (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 3. 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:

$$f^{\langle d,a,e,m,w \rangle}(\phi(s)) = [\![\rho(s) = d]\!] A_{z_1}(a, e, m, w, s),$$
$$g^{\langle d,a,e,m,w \rangle}(\phi(s)) = [\![\rho(s) = d]\!] A_{z_2}(a, e, m, w, s),$$

and writing $\vec{r_i}$ for the vector $\langle d_i, a_i, e_i, m_i, w_i \rangle$ in order to consider all ways to split the parameter values of x between the two children nodes, we can now compute

$$A_x(a, e, m, w, s) = \sum_{\vec{r_1} + \vec{r_2} = \langle \rho(s), a', e, m, w \rangle} (f^{\vec{r_1}} *_x^4 g^{\vec{r_2}})(\phi(s))$$

where $a' = a + |s_1^{-1}\{1_1, 1_2\} \cap s_2^{-1}\{1_1, 1_2\}|$. We point out that

$$\sum_{\vec{r_1} + \vec{r_2} = \langle \rho(s), a', e, m, w \rangle} (f^{\vec{r_1}} *_x^4 g^{\vec{r_2}})(\phi(s)) = 1$$

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 6, the parity of the number of marked-cc-solutions with $|E| - k$ edges and weight w is the same as the parity of the number of cc-candidates G' with $|E| - k$ edges, $|V(G')| - (|E| - k) - n_I(G')$ 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. Note that the degree-function is $\emptyset$ in this entry because there are no vertices contained in the root node by definition.

By Lemma 3, the time complexity of `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, as we only consider one instance of s per table entry, the complexity for each is $O^*(4^{tw})$. We point out that the size of the table is polynomial in n because there are a linear number of bags and a polynomial number of entries (combinations of parameters) for each bag. 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 `tw-copath` discussed above proves Theorem 4.

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

In order to use `tw-copath` to solve k-Co-Path 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.

Algorithm 1: Generating reduced instances

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

4.1 Kernelization and Branching

We start by describing our branching procedure `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 ℓ 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 `deg-branch` depends on both input parameters ℓ and D. We will select D to achieve the desired complexity in `copath` in Section 4.3. We also make use of a budget parameter b, which keeps track of how many more edges can be removed per the constraints of ℓ (b is initially set to ℓ).

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 a set of reduced instances which have had exactly ℓ edges removed. The size of the set 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). We now give an upper bound on the size of this set.

▶ **Lemma 8.** *Let T be a search tree formed by* `deg-branch`(G, ℓ, D, k, b). *The number of leaves of T is at most* $\binom{D+1}{2}^{\ell/(D-1)}$.

Proof of Lemma 8. The number of children of each interior node of T is $\binom{D+1}{2}$, resulting in at most $\binom{D+1}{2}^{depth(T)}$ leaves. The depth of T is limited by the second condition of the `if` on line 3 of Algorithm 1. For each recursive call, b is decremented by $(D-1)$, until $b \leq D-1$. As b is initially set to ℓ, this implies $depth(T) \leq \ell/(D-1)$, proving the claim. ◀

Finally, we argue that at least one member of the set of reduced instances returned by `deg-branch` is equivalent to the original. Consider a solution F to k-Co-Path Set in the original instance (G, k). Every vertex has at most two incident edges in $G[E \backslash F]$, and since all candidates are considered at every high-degree vertex, at least one branch correctly keeps all of these edges.

Table 2 Numerically obtained constants c_d, $3 \leq d \leq 17$, used in Lemma 9; originally given in Table 6.1 of [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	

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

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

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 in n.

Since the structure of k-Co-Path Set naturally provides some constraints on the degree sequence of yes-instances, we are able to apply Lemma 9 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.

▶ **Lemma 10.** *Let n_i be the number of vertices of degree i in a graph G for any $i \in \mathbb{Z}^+$, and Δ 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). ◀

▶ **Lemma 11.** *Let (G, k) be an instance of k-Co-Path Set such that G has n vertices and max degree at most $\Delta \in \{3, \ldots, 17\}$. If (G, k) is a yes-instance, then the treewidth of G is upper bounded by $k/3 + \epsilon n + c$, for any $\epsilon > 0$ and constant $c = n_\epsilon$ as defined in Lemma 9. A tree decomposition of the corresponding width can be constructed in polynomial time in n.*

Proof. Let n_ϵ be defined as in Lemma 9. Let G' be the graph formed by adding $N = n_\epsilon$ isolates to G. By Lemma 9, because G' has maximum degree at most Δ, $tw(G') \leq (1/6)n_3 + (1/3)n_4 + \ldots + c_\Delta n_\Delta + \epsilon(N + n)$. We can substitute the bound for n_3 from Lemma 10, which yields:

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

Algorithm 2: Deciding k-Co-Path Set

```
Algorithm copath ($G$,$k$)
    $(G', k')$ = 6k-kernel$(G, k)$
    for $k_1 \leftarrow 0$ to $k'$ do
        $Q_{k_1}$ = deg-branch$(G', k', k_1, 10, k_1)$
        foreach $(G_i, k_2) \in Q_{k_1}$ do
            if tw-copath$(G_i, k_2)$ then return true
    return false
```

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 Lemma 11 to reduced instances, computing the desired 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. Note that `deg-branch` considers *all* possible reduced instances, and thus if a (cc-)solution exists, it is contained in at least one reduced instance. 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 `tw-copath`, which correctly decides the problem with probability 2/3.

Proof of Theorem 1. We now analyze the running time of `copath`, as given in Algorithm 2. By Lemma 8, the size of each Q_{k_1} is $O(1.561^{k_1})$. For each reduced instance (G_i, k_2) in Q_{k_1}, we have $tw(G_i) \leq k_2/3 + \epsilon(6k) + c$ by Lemma 11.

Applying Theorem 4, `tw-copath` runs in time $O^*(4^{k_2/3+\epsilon 6k})$ for each reduced instance (G_i, k_2) in Q_{k_1} (with success probability at least 2/3). Each iteration of the outer `for` loop can then be completed in time

$$O^*(1.561^{k_1} 4^{k_2/3+\epsilon 6k}) = O^*(4^{k/3+\epsilon 6k}) = O^*(1.588^k),$$

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 `copath`. Additionally `copath` is linear-fpt, as the kernelization of [7] is $O(n)$, and the kernel has size $O(k)$, avoiding any additional poly(n) complexity from the `tw-copath` subroutine. Note that by Lemma 9 the tree decomposition can be found in polynomial time in the size of the reduced instance. Since reduced instances are subsets of $6k$-kernels, the linearity is unaffected because the graph has size polynomial in k. ◀

5 Conclusion

This paper gives an $O^*(4^{tw})$ 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 combined parameterization asking for a co-path set of size k resulting in ℓ disjoint paths is solvable in sub-exponential fpt time.

Acknowledgements. 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. `doi:10.1145/1250790.1250801`.

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.*, 2015. in press, DOI 10.1007/s10878-015-9901-y. URL: `http://dx.doi.org/10.1007/s10878-015-9901-y`.

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. `doi:10.1007/s00453-007-9133-3`.

9 S. Gaspers. *Exponential Time Algorithms – Structures, Measures, and Bounds*. VDM, 2010. URL: `http://bora.uib.no/bitstream/handle/1956/3436/Dr.thesis_Serge_Gaspers.pdf`.

10 T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. `doi:10.1007/BFb0045375`.

11 K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of STOC*, pages 345–354. ACM, 1987. `doi:10.1145/28395.383347`.

12 M. Pilipczuk. Solving connectivity problems parameterized by treewidth in single exponential time. In *MFCS*, pages 520–531, 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. *Journal of Combinatorial Optimization*, 27(1):3–13, 2014.