Fixed-Parameter Tractability of Maximum Colored Path and Beyond

FEDOR V. FOMIN, PETR A. GOLOVACH, and TUUKKA KORHONEN, Department of Informatics, University of Bergen, Bergen, Norway
KIRILL SIMONOV, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
GIANNOS STAMOULIS, LIRMM, Université de Montpellier, CNRS, Montpellier, France

We introduce a general method for obtaining fixed-parameter algorithms for problems about finding paths in undirected graphs, where the length of the path could be unbounded in the parameter. The first application of our method is as follows. We give a randomized algorithm, that given a colored n-vertex undirected graph, vertices s and t, and an integer k, finds an (s, t)-path containing at least k different colors in time $2^k n^{O(1)}$. This is the first FPT algorithm for this problem, and it generalizes the algorithm of Björklund, Husfeldt, and Taslaman on finding a path through k specified vertices. It also implies the first $2^k n^{O(1)}$ time algorithm for finding an (s, t)-path of length at least k. Our method yields FPT algorithms for even more general problems. For example, we consider the problem where the input consists of an n-vertex undirected graph G, a matroid M whose elements correspond to the vertices of G and which is represented over a finite field of order q, a positive integer weight function on the vertices of G, two sets of vertices $S, T \subseteq V(G)$, and integers p, k, w, and the task is to find p vertex-disjoint paths from S to T so that the union of the vertices of these paths contains an independent set of M of cardinality k and weight w, while minimizing the sum of the lengths of the paths. We give a $2^{p+O(k^2 \log(q+k))} n^{O(1)} w$ time randomized algorithm for this problem.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Fixed parameter tractability**;

Additional Key Words and Phrases: FPT algorithms, finding long paths, algebraic techniques in FPT

ACM Reference format:
Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. 2024. Fixed-Parameter Tractability of Maximum Colored Path and Beyond. *ACM Trans. Algor.* 20, 4, Article 36 (October 2024), 48 pages.
https://doi.org/10.1145/3674835

A preliminary version of this article appeared in the Proceedings of SODA 2023.
The research leading to these results has received funding from the Research Council of Norway via the project BWCA (Grant No. 314528). Kirill Simonov acknowledges support by the Austrian Science Fund (FWF, project Y1329). Giannos Stamoulis acknowledges support by the ANR project ESIGMA (ANR-17-CE23-0010) and the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).
Authors' Contact Information: Fedor V. Fomin, Department of Informatics, University of Bergen, Bergen, Norway; e-mail: fomin@ii.uib.no; Petr A. Golovach, Department of Informatics, University of Bergen, Bergen, Norway; e-mail: petr.golovach@uib.no; Tuukka Korhonen (Corresponding author), Department of Informatics, University of Bergen, Bergen, Norway; e-mail: tuukka.korhonen@uib.no; Kirill Simonov, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany; e-mail: kirillsimonov@gmail.com; Giannos Stamoulis, LIRMM, Université de Montpellier, CNRS, Montpellier, France; e-mail: giannos.stamoulis@lirmm.fr.

ACM 1549-6333/2024/10-ART36
https://doi.org/10.1145/3674835

1 Introduction

The study of long cycles and paths in graphs is a popular research direction in parameterized algorithms. Starting from the color-coding of Alon et al. [1], powerful algorithmic techniques have been developed [2, 3, 15, 16, 21, 29, 46, 49], see also [14, Chapter 10], for finding long cycles and paths in graphs. However, most of the known methods are applicable only in the scenario when the size of the solution is bounded by the parameter. Let us explain what we mean by that by the following example.

Consider two very related problems, k-Cycle and Longest Cycle. In both problems, we are given a graph G and an integer parameter k.[1] In k-Cycle we ask whether G has a cycle of length *exactly* k. In Longest Cycle, we ask whether G contains a cycle of length *at least* k. While in the first problem any solution should have exactly k vertices, in the second problem the solution could be even a Hamiltonian cycle on n vertices. The essential difference in applying color-coding (and other methods) to these problems is that for k-Cycle, a random coloring of the vertices of G in k colors will color the vertices of a solution cycle with different colors with probability e^{-k}. Such information about colorful solutions allows dynamic programming to solve k-Cycle (as well as the related k-Path problem, the problem of finding a path of length exactly k). However, since a solution cycle for Longest Cycle is not upper-bounded by a function of k, the coloring argument falls apart. As Fomin et al. write in [21] "This is why color-coding and other techniques applicable to k-Path do not seem to work here." Sometimes, like in the case of Longest Cycle, a simple "edge contraction" trick, see [14, Exercise 5.8], allows reducing the problem to k-Cycle. We are not aware of general methods for solving problems related to cycles and paths when the size of the solution is not upper-bounded by the parameter.

The main result of this article is a theorem that allows deriving algorithms for various parameterized problems about paths, cycles, and beyond, in the scenario when the size of the solution is not upper-bounded by the parameter. We discuss numerous applications of the theorem in the next section.

Our theorem is about finding a k-*colored* (S, T)-*linkage* in a colored graph. Let G be a graph, S and T be sets of vertices of G, and p be a positive integer. An (S, T)-*linkage* of order p is a set $\mathcal{P}$ of $p = |\mathcal{P}|$ vertex-disjoint paths, each starting in S and ending in T. The set of vertices in the paths of $\mathcal{P}$ is denoted by $V(\mathcal{P})$. The *total length* (or often just the length) of an (S, T)-linkage is the total number of vertices in its paths, i.e., $|V(\mathcal{P})|$. For a coloring $c : V(G) \to [n]$ of G, an (S, T)-linkage $\mathcal{P}$ is called k-*colored* if $V(\mathcal{P})$ contains at least k different colors, i.e., $|c(V(\mathcal{P}))| \geq k$. Let us note that in the above definition the sets S and T are not necessarily disjoint (in Section 4 we show how to reduce to the case that $|S| = |T| = p$ and S and T are disjoint) and that the coloring c is not necessarily a proper coloring in the graph-coloring sense. We also note that for vertices $s, t \in V(G)$, an $(\{s\}, \{t\})$-linkage of order 1 corresponds to an (s, t)-path. Our main result is the following:

Theorem 1.1. *There is a randomized algorithm, that given as an input an n-vertex graph G, a coloring $c : V(G) \to [n]$ of G, two sets of vertices $S, T \subseteq V(G)$, and integers p, k, in time $2^{k+p} n^{O(1)}$ either returns a k-colored (S, T)-linkage of order p and of the minimum total length, or determines that G has no k-colored (S, T)-linkage of order p.*

Few remarks are in order. First, Theorem 1.1 cannot be extended to directed graphs. It is easy to show, see Proposition 3.2, that finding a 2-colored (s, t)-path in a 2-colored directed graph is already NP-hard. Second, by another simple reduction, see Proposition 3.3, it can also be observed that if the time complexity of Theorem 1.1 could be improved to $(2 - \varepsilon)^{k+p} n^{O(1)}$ for $\varepsilon > 0$, even in the case when $p = 1$, G is colored with k colors, and $S = T = V(G)$, then Set Cover would admit a

[1]In this article, graphs are assumed to be undirected if it is not explicitly mentioned to be otherwise.

$(2-\varepsilon)^n(mn)^{O(1)}$ time algorithm, contradicting the **Set Cover Conjecture (SeCoCo)** of Cygan et al. [13]. We also remark that actually we prove an even more general result than Theorem 1.1, our result in full generality will be stated as Theorem 1.4. It can be also observed that by a simple reduction that subdivides edges, the coloring could be on the edges of G instead of vertices (or on both vertices and edges).

The algorithm in Theorem 1.1 invokes the DeMillo–Lipton–Schwartz–Zippel lemma for polynomial identity testing and thus is "heavily" randomized. We do not know whether Theorem 1.1 could be derandomized. The special case of Theorem 1.1 when the coloring is a bijection, the problem of finding an (S,T)-linkage of order p and of length at least k, can be reduced to the (rooted) topological minor containment. To see why, observe that if we enumerate all possible collections $\mathcal{P}$ of p paths of total length k, then we can check for each collection $\mathcal{P}$ if it is contained as a rooted topological minor in G. The topological minor containment admits a deterministic FPT algorithm parameterized by the size of the pattern graph [24]. However, the running time of the algorithm of Grohe et al. [24] is bounded by a tower of exponents in k and p. Our next theorem gives a deterministic algorithm for computing an (S,T)-linkage of order p and of length at least k whose running time is single-exponential in the the parameter k for any fixed value of p. The other advantage of the algorithm in Theorem 1.2 is that it works on directed graphs too. In the following statement, a *directed* (S,T)-linkage is defined analogously to an (S,T)-linkage but is composed of directed paths from S to T.

Theorem 1.2. *There is a deterministic algorithm that, given an n-vertex digraph G, two sets of vertices $S,T\subseteq V(G)$, an integer p, and an integer k, in time $p^{O(kp)}n^{O(1)}$ either returns a directed (S,T)-linkage of order p and of total length at least k, or determines that G has no directed (S,T)-linkage of order p and total length at least k.*

1.1 Applications of Theorem 1.1

Theorem 1.1 implies **fixed-parameter tractable (FPT)** algorithms for several problems. It encompasses a number of fixed-parameter-tractability results and improves the running times for several fundamental well-studied problems.

Longest path/cycle. When the coloring $c : V(G)\to[n]$ is a bijection, and thus all vertices of G are colored in different colors, then an (S,T)-linkage is k-colored if and only if its length is at least k. In this case, Theorem 1.1 outputs an (S,T)-linkage of order p with at least k vertices in time $2^{k+p}n^{O(1)}$. In particular, for $p=1$ it implies that Longest (s,t)-Path (i.e., for $s,t\in V(G)$ and $k\geq 0$, to decide whether there is an (s,t)-path of length at least k) is solvable in time $2^k n^{O(1)}$. Since one can solve Longest Cycle (to decide whether G contains a cycle of length at least k) by solving for every edge $st\in E(G)$ the Longest (s,t)-Path problem, Theorem 1.1 also yields an algorithm solving Longest Cycle in time $2^k n^{O(1)}$. To the best of our knowledge, the previous best known algorithm for Longest (s,t)-Path runs in time $4^k n^{O(1)}$ [22, 46] and the previous best known algorithm for Longest Cycle runs in time $1.66^{2k}n^{O(1)}=2.76^k n^{O(1)}$ [3, 49]. The former algorithm follows by combining the result of Fomin et al. [22] stating if an (s,t)-path of length exactly k can be found in $t(G,k)n^{O(1)}$ time, then Longest (s,t)-Path can be solved in $t(G,2k)n^{O(1)}$ time with the algorithm of Williams [46]. The latter algorithm follows by combining the result of Zehavi [49] stating that Longest Cycle is solvable in time $t(G,2k)n^{O(1)}$, where $t(G,k)$ is the best known running time for solving k-Path, with the fastest algorithm for k-Path of Björklund et al. [3].

For $p=2$, the problem of finding an (S,T)-linkage of length at least k is equivalent to the problem of finding a cycle of length at least k passing through a given pair of vertices s,t. A randomized algorithm of running time $(2e)^k n^{O(1)}$ for this problem, known as Longest (s,t)-Cycle, was given by Fomin et al. in [19, Theorem 4] (see also [20]).

As we already have mentioned the problem of finding an (S,T)-linkage of order p and of length at least k can be reduced to the (rooted) topological minor containment. For $p \geq 3$, Theorems 1.2 and 1.1 provide the first (randomized and deterministic) single-exponential in $k+p$ and single-exponential in k for constant p, respectively, algorithms for computing an (S,T)-linkage of order p and of length at least k. For directed graphs, Theorem 1.2 gives the first FPT algorithm for the problem parameterized by $k+p$.

T-cycle. In the T-Cycle problem, we are given a graph G and a set $T \subseteq V(G)$ of terminals. The task is to decide whether there is a cycle passing through all terminals [4, 28, 45]. By the celebrated result of Björklund et al. [4], T-Cycle is solvable in time $2^{|T|}n^{O(1)}$, and their algorithm in fact returns the shortest such cycle. To solve T-Cycle as an application of Theorem 1.1, we do the following. We pick a terminal vertex $t \in T$, create a twin vertex s of t (i.e., a vertex s with $N(s) = N(t)$), and color s and t with color 1. We then color all non-terminal vertices of G with color 1 too. The remaining terminal vertices $T \setminus \{t\}$ we color in $|T|-1$ colors from 2 to $|T|$, which ensures that no color repeats. Then G has a T-cycle if and only if there is a $|T|$-colored $(\{s\},\{t\})$-linkage of order 1. Therefore, using the algorithm of Theorem 1.1, we can also find the shortest T-cycle in time $2^{|T|}n^{O(1)}$. One could use Theorem 1.1 to generalize the algorithmic result of Björklund et al. in different settings. For example, instead of a cycle passing through all terminal vertices, we can ask for a cycle containing at least k terminals from a set T of unbounded size, in time $2^k n^{O(1)}$.

Another generalization of T-Cycle comes from covering terminal vertices by at most p disjoint cycles. For example, in the basic **vehicle routing problem (VRP)** one wants to route p vehicles, one route per vehicle, starting and finishing at the depot so that all the customers are supplied with their demands and the total travel cost is minimized [10]. In the simplified situation when the clients are viewed as terminal vertices T of a graph and routes in VRP are required to be disjoint, this problem turns into the problem of finding a "p-flower" of minimum total length containing all vertices of T. By p-flower we mean a family of p cycles that intersect only in one (depot) vertex s. To see this problem as a problem of finding a colored (S,T)-linkage, we replace the depot s by a set S of $2p$ vertices whose neighbors are identical to the neighbors of s. Then similar to T-Cycle, this variant of VRP reduces to computing a minimum length $(|T|+1)$-colored (S,S)-linkage of order p; thus it is solvable in time $2^{|T|+p}n^{O(1)}$ by Theorem 1.1.

Colored paths and cycles. The problems of finding a path, cycle, or another specific subgraph in a colored graph with the maximum or the minimum number of different colors appear in different subfields of algorithms, graph theory, optimization, and operations research [6, 8, 11, 12, 25, 31, 32, 42, 47]. In particular, the seminal color-coding technique of Alon et al. [1], builds on an algorithm finding a colorful path in a k-colored graph, that is, a path of k vertices and k colors, in time $O(2^k n)$.

In the Maximum Colored (s,t)-Path problem, we are given a graph G with a coloring $c : V(G) \to [n]$ and integer k. The task is to identify whether G contains a k-colored (s,t)-path, i.e., an (s,t)-path with at least k different colors. In the literature, this problem is also known as Maximum Labeled Path [12] and Maximum Tropical Path [11]. Theorem 1.1 yields the first FPT algorithm for Maximum Colored (s,t)-Path, as well as for Maximum Colored Cycle (decide whether G contains a k-colored cycle). It is also the first FPT algorithm for the even more restricted variant of deciding if a given k-colored graph contains any k-colored path. A recent article of Cohen et al. [11] claims a $O(2^k n^2)$ time deterministic algorithm for computing a shortest k-colored path in a given k-colored graph. Unfortunately, a closer inspection of the algorithm of Cohen et al. reveals that it computes a k-colored walk instead of a k-colored path.[2]

It is interesting to note that the minimization version of the colored (s,t)-path, i.e., to decide whether there is an (s,t)-path containing at most k different colors, is W[1]-hard even on very restricted classes of graphs [18].

Beyond graphs: frameworks. Frameworks provide a natural generalization of colored graphs. Following Lovász [35], we say that a pair (G,M), where G is a graph and $M = (V(G),\mathcal{I})$ is a matroid on the vertex set of G, is a *framework*. Then we seek for a path, cycle, or (S,T)-linkage in G maximizing the rank function of M. Note that frameworks (G,M) where M is a partition matroid generalize colored graphs. Indeed, the universe $V(G)$ of M is partitioned into color classes $L_1,\dots,L_n$ and a set I is independent if $|I \cap L_i| \leq 1$ for every color $i \in [n]$. However, by plugging different types of matroids into the definition of the framework, we obtain problems that cannot be captured by colored graphs.

Frameworks, under the name *pregeometric graphs*, were used by Lovász in his influential work on representative families of linear matroids [34]. The problem of computing maximum matching in frameworks is strongly related to the matchoid, the matroid parity, and polymatroid matching problems. See the *Matching Theory* book of Lovász and Plummer [36] for an overview. In their book, Lovász and Plummer use the term *matroid graph* for frameworks. In his most recent monograph [35], Lovász introduces the term frameworks, and this is the term we adopt in our work. More generally, the problems of computing specific subgraphs of large ranks in a framework belong to the broad class of problems about submodular function optimization under combinatorial constraints [7, 9, 40].

Let (G,M) be a framework and let $r\colon 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ be the rank function of the matroid M. The *rank of a subgraph* H of G is $r(V(H))$ and we denote it by $r(H)$. We say that an (S,T)-linkage $\mathcal{P}$ in a framework (G,M) is k*-ranked* if the rank of $\mathcal{P}$, that is the rank in M of the elements corresponding to the vertices of the paths of $\mathcal{P}$, is at least k. With additional work involving (lossy) randomized truncation of the matroid, it is possible to extend Theorem 1.1 from colored graphs to frameworks over a general class of representable matroids.

Theorem 1.3. *There is a randomized algorithm that, given a framework (G,M), where G is an n-vertex graph and M is represented as a matrix over a finite field of order q, sets of vertices $S,T \subseteq V(G)$, and an integer k, in time $2^{p+O(k^2\log(q+k))}n^{O(1)}$ either finds a k-ranked (S,T)-linkage of order p and of minimum total length, or determines that (G,M) has no k-ranked (S,T)-linkage of order p.*

With minor adjustments, Theorem 1.3 can be adapted for frameworks with matroids that are in general not representable over a finite field of small order. For example, uniform matroids, and more generally transversal matroids, are representable over a finite field, but the field of representation must be large enough. Despite this, we can apply Theorem 1.3 to transversal matroids. Similarly, it is possible to apply Theorem 1.3 in the situation when M is represented by an integer matrix over rationals with entries bounded by $n^{O(k)}$.

Weighted extensions. Theorem 1.1 can be extended into a weighted version in two different settings. The first setting is to have weights on edges that affect the length of the (S,T)-linkage. It is easy to see that by subdividing edges, coloring the subdivision vertices with a new "dummy color," and increasing k by one, all our algorithms work in the setting when the edges have polynomially bounded positive integer weights.

[2]The error in [11] occurs on p. 478. It is claimed that if P is a shortest (u,v)-path that uses the set C of colors and P' is a (w,t)-sub-path of P using colors $C' \subseteq C$, then P' must be a shortest (w,t)-path among all (w,t)-paths using colors C'. This claim is correct for walks but not for paths.

The second weighted extension is more interesting. It is to have weights on vertices and asking for an (S, T)-linkage containing a combination of weights and colors in a specific sense. In this setting, we have in addition to the coloring $c : V(G) \to [n]$ a weight function $\mathtt{we} : V(G) \to \mathbb{Z}_{\geq 1}$. For integers k, w, we say that an (S, T)-linkage $\mathcal{P}$ is (k, w)-*colored* if its vertices $V(\mathcal{P})$ contain a set $X \subseteq V(\mathcal{P})$ so that $|X| = k$, all vertices of X have different colors, and the total weight of X is exactly $\mathtt{we}(X) = \sum_{v \in X} \mathtt{we}(v) = w$. This weighted version does not follow by direct reductions but instead by a modification of Theorem 1.1 (in our main proof, we will directly prove Theorem 1.4 instead of Theorem 1.1).

Theorem 1.4. *There is a randomized algorithm that, given as an input an n-vertex graph G, a coloring* $c : V(G) \to [n]$ *of G, a weight function* $\mathtt{we} : V(G) \to \mathbb{Z}_{\geq 1}$*, two sets of vertices S, T, and three integers p, k, w, in time* $2^{k+p} n^{O(1)} w$ *either returns a* (k, w)*-colored* (S, T)*-linkage of order p and of minimum total length, or determines that no* (k, w)*-colored* (S, T)*-linkages of order p exist.*

Note that Theorem 1.4 implies Theorem 1.1 by setting all vertex weights to 1 and $w = k$. Theorem 1.4 allows to derive some applications of our technique that do not directly follow from Theorem 1.1, which we proceed to describe.

Longest T-cycle. Recall that in the T-Cycle problem the task is to find a cycle passing through a given set T of terminal vertices. Both the algorithm of Björklund et al. [4], and the application of the algorithm of Theorem 1.1 find in fact the shortest T-cycle. A natural generalization of the T-Cycle problem is the Longest T-Cycle problem, where in addition to the set T we are given an integer k and the task is to find a cycle of length at least k passing through the terminals T. Theorem 1.4 can be used to solve Longest T-Cycle in time $2^{\max(|T|,k)} n^{O(1)}$ as follows. First, if $|T| \geq k$, any T-cycle has length at least k and we just use the algorithm for T-Cycle. Otherwise, like in the reduction for T-Cycle, we first pick a terminal $t \in T$ and create a twin s of it. Then, we color s and t with color 1, and all the other vertices with different colors from 2 to n. We also assign weight 3 to the terminal vertices T, weight 1 to the vertex s, and weight 2 to all other vertices. We invoke Theorem 1.4 to find an $(\{s\}, \{t\})$-linkage of order 1 that contains a set X of vertices with distinct colors, size $|X| = k$, and weight $\mathtt{we}(X) = 2k + |T|$. Any such set X must be a superset of T and not contain s, and therefore the found path must correspond to a cycle of length at least k passing through the terminals T.

Vehicle routing with profits. With Theorem 1.4, we can give an algorithm for the VRP in a bit more general setting. In particular, we consider the situation where the depot has k parcels, p vehicles, and for each vertex v we know that we obtain a profit $\mathtt{we}(v)$ for delivering a parcel to that vertex. We can use Theorem 1.4 with the same reduction as used for VRP earlier, but instead letting the coloring of the vertices to be a bijection, to obtain a $2^{k+p} n^{O(1)} w$ time algorithm for determining the shortest routing by cycles intersecting only at the depot that yields a total profit of w.

Longest k-colored (S, T)*-linkage.* Theorem 1.4 can be also used to derive a longest path version of Theorem 1.1, in particular an algorithm that given a graph G, a coloring $c : V(G) \to [n]$, two sets of vertices $S, T \subseteq V(G)$, three integers k, p, ℓ, in time $2^{p+\ell+k} n^{O(1)}$ outputs a k-colored (S, T)-linkage of order p and length at least ℓ. The reduction is as follows. First, if $p \geq \ell$, then any (S, T)-linkage of order p has length at least ℓ, so we can use Theorem 1.1. Otherwise, we are looking for a k-colored (S, T)-linkage that contains at least $\ell - p$ edges. We subdivide every edge, and for each created subdivision vertex we assign a new color and weight $2k$. For the original vertices we keep their colors and assign weight 1. Now, any k-colored (S, T)-linkage of order p and length at least ℓ corresponds to an (S, T)-linkage of order p that contains a set X of vertices with distinct colors, size $|X| = k + \ell - p$, and weight exactly $\mathtt{we}(X) = (\ell - p) \cdot 2k + k$ (note that here we use the property that we are looking for an exact weight instead of maximum weight).

Weighted frameworks. We consider a generalization of frameworks into weighted frameworks. In particular, we say that a triple $(G, M, \mathtt{we})$, where G is a graph, $M = (V(G), \mathcal{I})$ is a matroid, and $\mathtt{we} : V(G) \to \mathbb{Z}_{\geq 1}$ is a weight function, is a *weighted framework*. Now we can say that an (S,T)-linkage $\mathcal{P}$ in a weighted framework $(G, M, \mathtt{we})$ is (k,w)*-ranked* if $V(\mathcal{P})$ contains a set X of vertices with $X \in \mathcal{I}$, size $|X| = k$, and weight $\mathtt{we}(X) = w$. By using the same reduction as from Theorem 1.1 to Theorem 1.3, we obtain the following theorem.

Theorem 1.5. *There is a randomized algorithm that given a weighted framework $(G, M, \mathtt{we})$, where G is an n-vertex graph and M is represented as a matrix over a finite field of order q, sets of vertices $S, T \subseteq V(G)$, and integers p, k, w, in time $2^{p+O(k^2 \log(q+k))} n^{O(1)} w$ either finds a (k,w)-ranked (S,T)-linkage of order p and of minimum total length, or determines that $(G, M, \mathtt{we})$ has no (k,w)-ranked (S,T)-linkages of order p.*

Note that Theorem 1.5 implies Theorem 1.3 by setting all vertex weights to 1 and setting $w = k$.

Finally, we remark that even though the correctness argument of our algorithm is technical, the algorithm itself is simple and practical, consisting of only simple dynamic programming over walks in the graph. In particular, the observed practicality of the algorithm of Björklund et al. [4] for T-Cycle on graphs with thousands of vertices holds also for our algorithm.

Organization of the article. The rest of the article is organized as follows. In Section 2 we overview our techniques and outline our algorithms. In Section 3 we recall definitions and preliminary results. In Section 4 we prove the main result, i.e., Theorem 1.4 (recall that Theorem 1.4 implies Theorem 1.1). In Section 5, we give the extensions of our results from colored graphs to frameworks, i.e., Theorem 1.5. In Section 6, we prove Theorem 1.2. Finally, we conclude in Section 7.

2 Techniques and Outline

The techniques behind Theorem 1.1 build on the idea of exploiting cancellation of monomials, a fundamental tool in the area [2–5, 29, 30, 33, 42, 46]. In particular, we build on the cycle-reversal-based cancellation for T-Cycle introduced by Björklund et al. [4], and on the bijective labeling-based cancellation introduced by Björklund [2] (see also [3]). The algorithm of Theorem 1.2 builds on color-coding [1], generalizing ideas that appeared in [19] for finding an (s,t)-cycle of length at least k.

In Section 2.1, we highlight new ideas of the techniques behind Theorem 1.1 in comparison to the earlier works. In Section 2.2, we give a more detailed outline of the proof of Theorem 1.1, and in Section 2.3, we give an outline of the proof of Theorem 1.2.

2.1 New Techniques for Theorem 1.1

Let us first focus on the single path case of Theorem 1.1, i.e., $p = |S| = |T| = 1$, corresponding to the question of finding a k-colored (s,t)-path. Our algorithm is analogous to the algorithm of Björklund et al. [4] for T-Cycle, but instead of having the "interesting set" of vertices T fixed in advance, our algorithm can choose any interesting set $X \subseteq V(G)$ of vertices of size $|X| = k$ included in the path "on the fly" in the dynamic programming over the walks. In particular, our dynamic programming over walks can choose whether it gives a label to a vertex or not. This is the crucial difference to the earlier works where there would be a set of vertices $Y \subseteq V(G)$ fixed in advance so that a vertex of Y would always be given a label if encountered in the walk and the vertices $V(G) \setminus Y$ would never be given labels [2–4, 42]. This would impose a limitation that because these algorithms work in time exponential in the number of labels used (i.e., 2^k, where k is the number of labels), the intersection of the found path with the set Y would have to be bounded in the parameter. This explains why the previous techniques could not yield an FPT-algorithm for Maximum Colored Path, as no such suitable set Y can be fixed in advance.

Our on the fly labeling of vertices allows our algorithm to find paths that visit the same color multiple times, while still making sure that at least k different colors are visited. In particular, the interesting set $X \subseteq V(G)$ of vertices in the path that we want to label is any set of size k that contains k different colors. While our dynamic programming is still a straightforward dynamic programming over walks, the main difficulty over previous works is the argument that if no solution exists, then the polynomial that we compute is the zero polynomial, i.e., all unwanted walks cancel out.

First, the argument of cancellation in the case when two vertices of the same color are given a label is a now-standard application of the bijective labeling based cancellation of Björklund [2]. Therefore, our main focus is on a cancellation argument for walks that do not form a path and k vertices of different colors have been labeled. Here, our starting point is the cycle reversal based cancellation argument for T-Cycle [4], but in our case significantly more arguments are needed. In particular, the main difference to earlier works caused by the introduction of the on the fly labeling is that a vertex can occur in a walk as both labeled and unlabeled. Very much oversimplified, this case is handled by a new label-swap cancellation argument, where a label is moved from a labeled occurrence of a vertex into an unlabeled occurrence of the vertex. While in isolation this argument is simple, it causes significant complications when combining with the cycle reversal based cancellation, in particular because of the "no labeled digons" property we have to impose to the labeled walk. However, we manage to combine these two arguments into a one very technical cancellation argument.

Then, let us move from one (s, t)-path to an (S, T)-linkage. This generalization of using cancellation of monomials to find multiple paths is foreshadowed by an algorithm for minimum cost flow by Lingas and Persson [33]. However, their arguments are considerably simpler due to not having labels on the walks.

To find (S, T)-linkages, we use a similar dynamic programming to the one path case, extending the set of walks from S to T one walk at a time. Here, we must introduce a new cancellation argument for the case when two different walks intersect. This argument is again simple in isolation: take the intersection point of the two intersecting walks and swap the suffixes of them starting from this point. First, to make sure that this operation does anything we need to make sure that the suffixes are not equal. We do this by enforcing that the ending vertices of the walks are different already in the dynamic programming, which adds the extra 2^p factor to the time complexity. The second complication is that again, this suffix swap operation does not play well together with the other cancellation arguments, and we need to again significantly increase the complexity of the combination of the three cancellation arguments. In the end, we have to consider 18 different cases in our cancellation argument, see Definition 4.9.

The extension from Theorem 1.1 to the weighted setting of Theorem 1.4 is a simple modification of the dynamic programming so that also the weight of the labeled vertices X is stored. Interestingly, this argument could be extended in principle to look for paths containing a set of vertices X with any property of X that could be efficiently evaluated in dynamic programming.

2.2 Outline of Theorem 1.1

We first give the outline of the algorithm for the single path case of finding a k-colored (s, t)-path, and then discuss the generalization to (S, T)-linkage.

Superficially, our approach follows the one of Björklund et al. [4] developed for the T-cycle problem. Similar to Björklund et al., for every length $\ell \geq 1$, we define a certain family of walks C_ℓ

and a polynomial $f(C_\ell)$ so that over a finite field of characteristic 2, the polynomial $f(C_\ell)$ is not the zero polynomial if the graph contains a k-colored (s,t)-path of length ℓ, and the polynomial $f(C_\ell)$ is the zero polynomial if the graph does not contain any k-colored (s,t)-path of length at most ℓ. Then by making use of the DeMillo–Lipton–Schwartz–Zippel lemma [44, 50], finding a k-colored (s,t)-path of minimum length boils down to evaluating the polynomial $f(C_\ell)$ for a uniformly random assignment of values to the variables for increasing ℓ.

With k-colored path, the role similar to the role of terminal vertices in T-cycle is played by a subset X of k vertices of the path with k different colors. However, a priori we do not know this set X, and there could be n^k candidates so we cannot enumerate them. Because of that, we define the polynomial f on families of *labeled*(s,t)-walks in the graph G. A labeled (s,t)-walk of length ℓ is a pair of sequences $W = ((v_1,\dots,v_\ell),(r_1,\dots,r_\ell))$, where $v_1,\dots,v_\ell$ is an (s,t)-walk of length ℓ, and $r_1,\dots,r_\ell$ is a sequence of numbers from $[0,k]$ indicating a labeling. The interpretation of the labeling is that $r_i = 0$ indicates that the index i of the walk is not labeled, and $r_i \geq 1$ indicates that the index i is labeled with the label r_i, with the interpretation that the vertex v_i at this index is selected to the set X. We require the labeling to be *bijective*, meaning that each label from $[k]$ is used exactly once in the walk, but the "non-label" 0 can repeat multiple times.

Next, we present the definition of the polynomial f. We will denote the coloring of the input graph G by the function $c : V(G) \to [n]$. The polynomial f is over $\mathrm{GF}(2^{3+\lceil \log_2 n \rceil})$, which is a field of characteristic 2 and order $\geq 8n$. With every edge $uv \in E(G)$ we associate a variable $f_E(uv)$, with every vertex $v \in V(G)$ we associate a variable $f_V(v)$, and with every color-label pair $(x,y) \in [n] \times [k]$ we associate a variable $f_C(x,y)$. Here, the purpose of the subscripts V, E, and C is distinguish the "vertex variables" f_V, "edge variables" f_E, and "color-label pair variables" f_C from each other. For a labeled walk $W = ((v_1,\dots,v_\ell),(r_1,\dots,r_\ell))$ we associate the monomial

$$f(W) = \prod_{i=1}^{\ell-1} f_E(v_i v_{i+1}) \cdot \prod_{i\in[\ell]\,:\, r_i\neq 0} f_V(v_i)\cdot f_C(c(v_i), r_i).$$

Because W has length ℓ and the labeling is bijective, $f(W)$ has degree $\ell - 1 + 2k$, being a product of $\ell - 1$ edge variables, k vertex variables, and k color-label pair variables. For the family of walks C_ℓ, which we will define immediately, we are interested in the polynomial

$$f(C_\ell) = \sum_{W\in C_\ell} f(W).$$

For vertices s,t and integers k,ℓ, the family C_ℓ is the family of all labeled (s,t)-walks

$$W = ((v_1 = s, v_2, \dots, v_\ell = t),(r_1,\dots,r_\ell)),$$

of length ℓ, where the labeling is bijective, and which do not contain "labeled digons." By a *labeled digon* we mean a subwalk $v_{i-1}v_iv_{i+1}$ with $v_{i-1} = v_{i+1}$ and v_i being a labeled vertex (i.e., $r_i \neq 0$). It is not immediately clear that having no labeled digons is useful, but this will turn out to be crucial similarly to the property of having no T-digons in the algorithm for T-cycle [4].

It is not difficult to prove that when a graph has a k-colored (s,t)-path of length ℓ, then $f(C_\ell)$ is not the zero polynomial. Indeed, a path has no repeated vertices and thus has no labeled digons, so if we take a k-colored (s,t)-path $v_1,\dots,v_\ell$ and let the labels $r_1,\dots,r_\ell$ take the values from $[k]$ on k vertices with k different colors, then the labeled walk $W = ((v_1,\dots,v_\ell),(r_1,\dots,r_\ell))$ appears in C_ℓ, and thus a corresponding monomial $f(W)$ appears in $f(C_\ell)$. Because $v_1,\dots,v_\ell$ is a path and the labeled vertices have different colors, we can recover the labeled walk W uniquely from the monomial $f(W)$, and therefore the monomial $f(W)$ must occur exactly once in the polynomial $f(C_\ell)$ (i.e., with coefficient 1), and therefore $f(C_\ell)$ is not the zero polynomial.

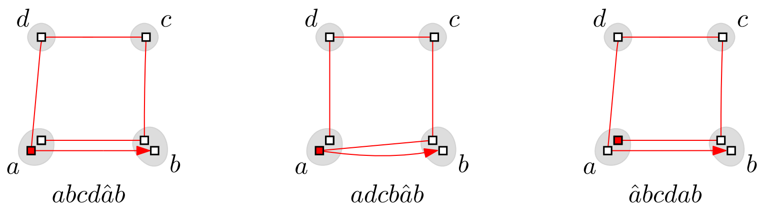


Fig. 1. An illustration of the walks $abcd\hat{a}b$, $a\overleftarrow{bcd}\hat{a}b = adcb\hat{a}b$, and $\hat{a}bcdab$. The gray bags correspond to vertices of the graph. The squares are copies of the corresponding bag-vertex and these together with the red path illustrate the order the vertices appear in the walk. Red squares correspond to labeled vertices.

The proof of the opposite statement—absence of a k-colored (s,t)-path of length $\leq \ell$ implies that $f(C_\ell)$ is the zero polynomial—is more complicated. We have to show that in this case each monomial $f(W)$ for labeled walks $W \in C_\ell$ occurs an even number of times in the polynomial $f(C_\ell)$, in particular that there is an even number of labeled walks $W \in C_\ell$ for every monomial $f(W)$. The proof is based on constructing an *f-invariant fixed-point-free involution* ϕ on C_ℓ, that is, a function $\phi : C_\ell \to C_\ell$ such that for every $W \in C_\ell$ it holds that (1) $f(W) = f(\phi(W))$ (f-invariant), (2) $\phi(W) \neq W$ (fixed-point-free), and (3) $\phi(\phi(W)) = W$ (involution).

Let us start with the easy part of the proof, that is, constructing such ϕ for labeled walks where two vertices of the same color are labeled (which could be two different occurrences of the same vertex). In this case, let $1 \leq i < j \leq \ell$ be the lexicographically smallest pair of indices so that $c(v_i) = c(v_j)$, $r_i \neq 0$, and $r_j \neq 0$. The function ϕ works by swapping r_i with r_j. Because each label from $[k]$ occurs in $r_1, \ldots, r_\ell$ exactly once, in particular $r_i \neq r_j$, this results in a different labeled walk $\phi(W)$ with the same monomial $f(\phi(W)) = f(W)$, and moreover $W = \phi(\phi(W))$ holds. After this argument, we can let $C_\ell^* \subseteq C_\ell$ be the family of labeled walks in C_ℓ where all labeled vertices have different colors, and we know that $f(C_\ell^*) = f(C_\ell)$. Therefore, it suffices to construct an f-invariant fixed-point free involution $\phi : C_\ell^* \to C_\ell^*$.

Now, the first approach would be to adapt the strategy of Björklund et al. for our purposes. This will not directly work but in the end we will make a considerable generalization of their approach to work for our purposes. The essence of their strategy is the following. Since walks from C_ℓ^* do not have labeled digons and because there is no k-colored (s,t)-path of length $\leq \ell$, it is possible to show that every walk $W \in C_\ell^*$ has a "loop," that is a subwalk vUv starting and ending in the same vertex v, and so that U is not a palindrome. Then $\phi(W)$ is the walk W' obtained from W by reversing U. This approach does not work directly in our case. The reason is that a labeled vertex could also occur several times in a walk as unlabeled. Because of that reversing a subwalk can result in a walk with a labeled digon, and thus ϕ could map W outside the family C_ℓ^*. For example, for a walk $abcd\hat{a}b$ (here $\hat{a}$ is a labeled vertex), reversing $a\overleftarrow{bcd}\hat{a}b$ results in walk $adcb\hat{a}b$ with labeled digon $b\hat{a}b$.

A natural "patch" for that type of walks that we introduce (which also will not directly work but gets us closer to a working ϕ) is to not reverse in this kind of situation but to apply a new type of operation of swapping a label from one occurrence of a vertex to another occurrence of it. For example, swapping a label for $abcd\hat{a}b$ would result in $\hat{a}bcdab$. This results in a different labeled walk contributing the same monomial $f(W)$ to the polynomial. See Figure 1 for an illustration of the above examples.

However, the new operation of swapping a label brings us new challenges. First of all, swapping a label could again result in a labeled digon. For example, swapping a label for walk $\hat{a}bcac$ results in walk $abc\hat{a}c$ with labeled digon $c\hat{a}c$. An attempt to "patch" this by using a "mixed" strategy—when

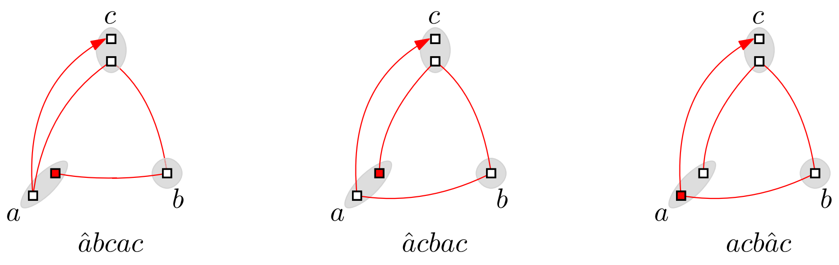


Fig. 2. An illustration of the walks $\hat{a}bcac$, $\overleftarrow{\hat{a}bcac} = \hat{a}cbac$, and $acb\hat{a}c$. The gray bags correspond to vertices of the graph. The squares are copies of the corresponding bag-vertex and these together with the red path illustrate the order the vertices appear in the walk. Red squares correspond to labeled vertices.

possible, swap a label, otherwise reverse—does not work either. For example, for walk $W = \hat{a}bcac$ we cannot label swap (that will result in a labeled digon $c\hat{a}c$), hence we reverse. Thus we obtain walk $W' = \phi(W) = \overleftarrow{\hat{a}bcac} = \hat{a}cbac$. For W', swapping a label for a is a valid operation, thus $\phi(W') = acb\hat{a}c$, but then we would have that $\phi(\phi(W)) \neq W$. See Figure 2 for an illustration of the above example.

The situation appears desperate: introducing more patches to the strategy seems to bring us even more problems. However, a bit surprisingly, in the end we manage to define the function ϕ by using a delicate strategy on when a subwalk reversal could be applied and when a label swap could be applied. The situation is made even more complicated as ϕ has to be defined recursively in order to deal with palindromic loops. All of Section 4.4 is devoted to defining this strategy (for the more general setting of (S, T)-linkages) and to the proof of its correctness.

To evaluate the polynomial $f(C_\ell)$, we apply quite standard dynamic programming techniques. In particular, the polynomial can be evaluated in $2^k n^{O(1)}$ time by dynamic programming over walks, where we store the length of the walk, the last two vertices of the walk, the subset of labels used so far (causing the 2^k factor), and whether the last vertex is labeled. This is similar to the dynamic programming for T-CYCLE [4], with the difference only in that it is chosen in the dynamic programming which vertices of the walk are labeled, and that instead of a subset of T we store the subset of the labels.

To extend the algorithm from a single (s, t)-path to an (S, T)-linkage of order p, we define a family C_ℓ of *labeled walkages* and a polynomial $f(C_\ell)$ over them. We note that by a simple reduction we can assume that $|S| = |T| = p$, and that S and T are disjoint. A labeled walkage of order p and total length ℓ is a p-tuple $\mathcal{W} = (W^1, \ldots, W^p)$ of labeled walks W^i, whose sum of the lengths is ℓ. The family C_ℓ contains labeled walkages $\mathcal{W}$ with the following properties: They have order p, total length ℓ, the starting vertices are ordered according to a total order on $V(G)$, ending vertices are distinct (each vertex in T is an ending vertex of exactly one walk in $\mathcal{W}$), the labeling is bijective (each label from $[k]$ is used exactly once), and no walk in $\mathcal{W}$ contains a labeled digon.

The monomial $f(\mathcal{W})$ is then defined as

$$f(\mathcal{W}) = \prod_{i=1}^{p} f(W^i),$$

and the polynomial $f(C_\ell)$ as

$$f(C_\ell) = \sum_{\mathcal{W} \in C_\ell} f(\mathcal{W}).$$

The definitions are analogous to the single path case, in particular, we recover the previously explained single path case by setting $p = 1$. The proof that if there exists a k-colored (S, T)-linkage of order p and total length ℓ then $f(C_\ell)$ is not the zero polynomial is directly analogous to the one path case. Also the proof that we can consider the smaller family $C_\ell^* \subseteq C_\ell$ where all labeled vertices have different colors is analogous.

However, to prove that if there is no k-colored (S, T)-linkages of order p and total length $\leq \ell$ then $f(C_\ell^*)$ is the zero polynomial we need new cancellation arguments beyond the previous cycle reversal and label swap arguments. In particular, none of the previously considered arguments can be applied if we have a labeled walkage $\mathcal{W} = (W^1, W^2)$ of order two, where both W^1 and W^2 are labeled paths that intersect. In this case, the new argument is that we could swap the suffixes of W^1 and W^2 starting from the intersection point. For example, for a walkage $\mathcal{W} = (abc\hat{d}e, xycuv)$, we define $\phi(\mathcal{W}) = (abcuv, xyc\hat{d}e)$. The property that the walks in $\mathcal{W}$ have different ending vertices is crucial here to ensure that $\phi(\mathcal{W}) \neq \mathcal{W}$.

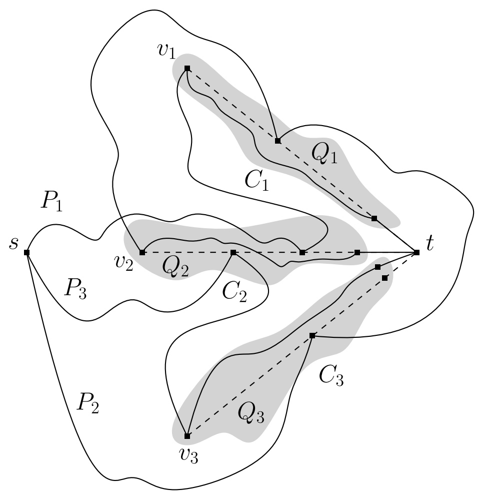
However, with this suffix swap cancellation argument we run also into new challenges. In particular, the first problem is that the suffix swap could create labeled digons, for example, when $\mathcal{W} = (ab\hat{c}de, xycbu)$ both of the walks are paths, but swapping the suffix after c would create a labeled digon. In this situation, we can instead use the label swap operation on c, from the first walk to the second, but of course this will add again even more complications. In the end, we manage to extend the strategy of ϕ from paths to linkages, but it makes the definition of ϕ even more complicated (see Definition 4.9, the path case uses the case groups A and C, while the linkage case needs the addition of case groups B and D).

The dynamic programming for (S, T)-linkage is similar to the (s, t)-path, extending the walks in the walkage one walk at the time. It requires two new fields to store, the index of the walk that we are currently extending, and the subset of the ending vertices T that have been already used. Storing the used ending vertices causes the additional factor 2^p in the time complexity (as we can assume that $|T| = p$).

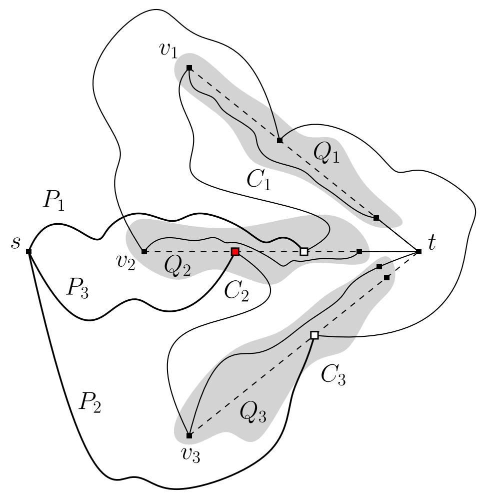
2.3 Outline of Theorem 1.2

Recall that the main difference to Theorem 1.1 is that Theorem 1.2 provides a deterministic algorithm that, moreover, works on directed graphs. The price is, however, that this algorithm is only suitable for the special case of finding an (S, T)-linkage of total length at least k rather than a k-colored one, and the time complexity as a function of k and p is higher. Theorem 1.2 thus requires a completely different toolbox: the algorithm is based on ideas of random separation, which is a technique that allows for efficient derandomization. Our result can be seen as a generalization of earlier works on finding paths and cycles of length at least k, the closest one being the result of Fomin et al. [19] on finding an (s, t)-cycle of length at least k. Note that their result is stated for undirected graphs, and that the problem of finding an (S, T)-linkage of order 2 and total length at least k is equivalent to the problem of finding an (s, t)-cycle of length at least k on undirected graphs. (The equivalence is up to increasing k by 2, since we need to introduce twins of s and t in order to preserve vertex-disjointness of paths in the target (S, T)-linkage.) Also, closely related is the result of Zehavi [49] for finding directed cycles of length at least k, which can be reduced to finding a directed (s, t)-path of length at least k by guessing an edge ts on the cycle, which, in turn, is exactly the problem of finding a directed linkage of order 1 and length at least k.

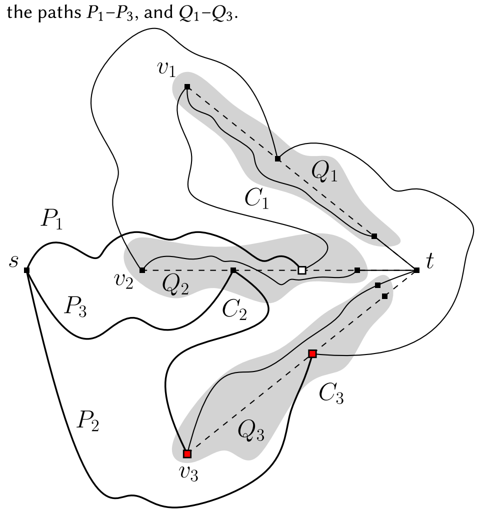
Similarly to the earlier results, the case where the target (S, T)-linkage is of length close to k can be covered by a standard application of color-coding [1]; in this way it is even possible to identify an (S, T)-linkage with length of order pk in the desired running time of Theorem 1.2. The difficulty is that the length of the (S, T)-linkage can be arbitrarily larger than k or pk. While because of



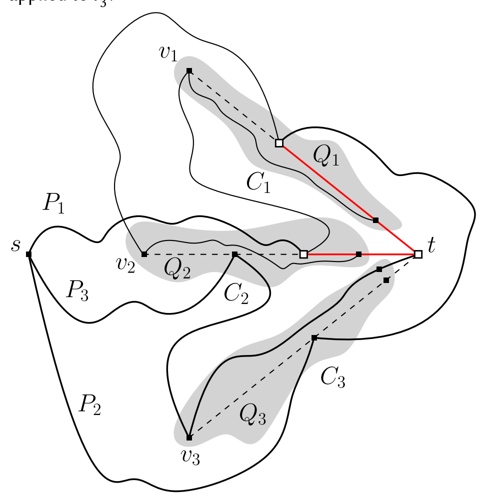
(a) The given digraph G, edge directions are implicit along the paths P_1–P_3, and Q_1–Q_3.



(b) Starting state S^1: two tokens belong to Q_2 so **Push** is applied to t_3^1.



(c) In S^2, $t_3^2 = v_3$ so **Clear** is applied to Q_3, moving t_3^2 to t and t_2^2 next along P_2.



(d) Finishing state S^3: P_2 is continued via Q_1, P_1 via Q_2, and P_3 is preserved to obtain the solution.

Fig. 3. Illustration to the proof of Lemma 6.4. Large empty squares mark tokens in the current state; those with red filling are moved by the next rule.

that it would be intractable to highlight the target (S, T)-linkage as a whole, it is still possible to apply random separation to give distinct colors to k-length segments at the end of each path in the (S, T)-linkage. The main hurdle is then to argue that at least in one color we can pick a finishing segment as an arbitrary shortest path of length k, without intersecting any other path in an optimal solution. Afterward, finding the desired (S, T)-linkage is easy, as the length requirement is already satisfied; one only needs to find a suitable connection to complete the (S, T)-linkage, which exists as witnessed by the optimal solution.

Lemma 6.1 encapsulates the novel combinatorial result allowing the approach above, strongly generalizing a similar basic idea that appeared in [19] for two undirected paths. To give an intuition behind the lemma (see also Figure 3), observe first that the problem of finding an (S, T)-linkage of order p and of total length at least k is equivalent to the problem of finding an (s,t)-*linkage* of order p and of total length at least $k+2$, where an (s,t)-linkage of order p consists of p internally disjoint (s,t)-paths, for some $s,t \in V(G)$. Now let (s,t)-paths $P_1,\ldots, P_p$ come from the shortest solution, i.e., an (s,t)-linkage of order p and the smallest total length which is at least k. Let the sets $C_1,\ldots, C_p$ be the result of random separation applied to k-length suffixes of $P_1 - \{t\},\ldots, P_p - \{t\}$, i.e., for each $i \in [p]$, the k-length suffix of $P_i - \{t\}$ is contained in C_i. The algorithm of Theorem 1.2 seeks to find a solution where for some $i \in [p]$, vertex v_i is the kth vertex of P_i from t, and Q_i is a k-length shortest path from v_i to t inside C_i, by guessing $v_i \in C_i$ and taking an arbitrary path Q_i of the form above. The solution is then any collection of an (s,v_i)-path and $p-1$ many (s,t)-paths that do not intersect each other and Q_i, together with Q_i. If Q_i does not intersect the (s,v_j)-prefix of P_j, for each $j \in [p]$, then the paths $P_1,\ldots, P_p$ certify that the desired collection of paths exists. Now comes Lemma 6.1: it claims, roughly, that if this is not the case for all $i \in [p]$, then there is a shorter (s,t)-linkage given by prefixes of $P_1,\ldots, P_p$ and suffixes of $Q_1,\ldots, Q_p$ (introducing another color to the random separation makes sure that the total length of the prefixes is still at least k), which is a contradiction.

The proof of Lemma 6.1 can be imagined as the following token sliding game. First, we put a token on each P_i, at the first place of intersection with some Q_j. Then we move the tokens by applying two rules, *Push* and *Clear*. If two tokens end up on the same Q_j for some $j \in [p]$, we move the farthest of them from t further along its path P_i, until it hits another $Q_{j'}$; this is called *Push*. As for the *Clear*, if at any step h the current token t_i^h of the path P_i reaches the vertex v_i, we forfeit this path: the token is moved to t, which corresponds to the ith path of the shorter solution being exactly P_i, and all other tokens on Q_i are moved next along their paths similarly to the rule *Push*. Moreover, every future application of any rule will not place a token on Q_i, skipping it to the next Q_j that is still active. Clearly, this game is finite, as tokens are only being slid further along their paths. The main claim of Lemma 6.1 is that when the game is over, there is at least one remaining active token, these tokens are one per a path in $\{Q_j\}_{j\in[p]}$ (since *Push* is not applicable), and that all corresponding paths P_i can be simultaneously extended each along its own Q_j instead of taking their original routes, without intersections with the previously fixed paths (since in *Push* we always keep the closest token to t). This is a shorter solution since a token of P_i, if active, is inside some Q_j at distance less than k from t, and the prefix of P_i up to this token is shorter than the prefix of P_i up to v_i.

Another challenge the proof of Theorem 1.2 faces, is that while the random separation approach is well-known, it is normally applied to separating two, rarely three (e.g., [19]), sets. We, on the other hand, need to apply random separation to p sets simultaneously, while making sure that it can be derandomized. To this end, in Lemma 6.4, we devise in a deterministic way a family of functions that models random separation of p sets of size at most k each. The size of this family is bounded by $p^{O(kp)} \log n$, which matches the inverse probability (up to the $\log n$ factor) of coloring the universe in p colors uniformly at random so that each set receives its own color. The construction is based on perfect hash families [39].

3 Preliminaries

In this section, we introduce basic notation and state some auxiliary results.

3.1 Basic Definitions and Preliminary Results

We use $\mathbb{Z}_{\geq 1}$ to denote the set of positive integers and $\mathbb{Z}_{\geq 0}$ the set of non-negative integers. Also, given integers p, q such that $p < q$, we use $[p,q]$ to denote the set $\{p, p+1, \ldots, q\}$ and, if $p \geq 1$, we use $[p]$ to denote the set $\{1,\ldots,p\}$.

Parameterized Complexity. We refer to the book of Cygan et al. [14] for introduction to the area. Here, we only briefly mention the notions that are most important to state our results. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is a set of strings over a finite alphabet Σ. An input of a parameterized problem is a pair (x, k), where x is a string over Σ and $k \in \mathbb{N}$ is a *parameter.* A parameterized problem is FPT if it can be solved in time $f(k) \cdot |x|^{O(1)}$ for some computable function f. The complexity class FPT contains all FPT parameterized problems.

Graphs. We use standard graph-theoretic terminology and refer to the textbook of Diestel [17] for missing notions. We consider only finite graphs, and the considered graphs are assumed to be undirected if it is not explicitly said to be otherwise. For a graph G, $V(G)$ and $E(G)$ are used to denote its vertex and edge sets, respectively. Throughout the article we use $n = |V(G)| = |G|$ and $m = |E(G)|$ if this does not create confusion. For a graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X. For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v, i.e., the set of vertices that are adjacent to v in G. For $X \subseteq V(G)$, $N_G(X) = \left(\bigcup_{v \in X} N_G(v)\right) \setminus X$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. If G is a digraph, $N_G^+(v)$ denotes the *out-neighborhood* of v, i.e., the set of vertices that are adjacent to v in G via an arc from v, and $N_G^-(v)$ is the *in-neighborhood*, defined symmetrically for arcs going to v. We may omit subscripts if the considered graph is clear from a context.

A walk W of length ℓ in G is a sequence of vertices $v_1, v_2, \ldots, v_\ell$, where $v_i v_{i+1} \in E(G)$ for all $1 \leq i < \ell$. The vertices v_1 and v_ℓ are the *endpoints* of W and the vertices $v_2, \ldots, v_{\ell-1}$ are the *internal* vertices of W. A path is a walk where no vertex is repeated. For a path P with endpoints s and t, we say that P is an (s, t)-path. A cycle is a path with the additional property that $v_\ell v_1 \in E(G)$ and $\ell \geq 3$.

DeMillo–Lipton–Schwartz–Zippel Lemma. Our strategy involves the use the DeMillo–Lipton–Schwartz–Zippel lemma for randomized polynomial identity testing.

Lemma 3.1 ([44, 50]). *Let p be a polynomial on variables $x_1, \ldots, x_r$, of total degree d over a field $\mathbb{F}$ that is not the zero polynomial, and let S be a subset of $\mathbb{F}$. If each x_i is independently assigned a uniformly random value from S, then the value of $p(x_1, \ldots, x_r)$ is zero with probability at most $d/|S|$.*

3.2 Hardness Results

We conclude this section by showing the NP-hardness of finding a k-colored (s, t)-path on directed graphs, for any $k \geq 2$, and the optimality of the time complexity of Theorem 1.1 assuming the SeCoCo of Cygan et al. [13].

We start with the hardness for directed graphs.

Proposition 3.2. *For any integers $k, \ell \geq 2$, it is* NP*-complete to decide, given a directed graph G, a coloring $c\colon V(G) \to [\ell]$, and two vertices s and t, whether G has a k-colored (s, t)-path.*

Proof. We show the claim for $k = \ell = 2$ as it is straightforward to generalize the proof for other values of k and ℓ. We reduce from the Disjoint Paths problem on directed graphs. The task of this problem is, given a (directed) graph G and k pairs of terminal vertices (s_i, t_i) for $i \in \{1, \ldots, k\}$, decide whether G has vertex-disjoint (s_i, t_i)-paths for $i \in \{1, \ldots, k\}$. This problem is well-known to be NP-complete on directed graphs even if $k = 2$ [23]. Consider an instance $(G, (s_1, t_1), (s_2, t_2))$ of Disjoint Paths, where G is a directed graph. We assume that the terminal vertices are pairwise distinct. We construct the directed graph G' from G by adding a vertex w and arcs (t_1, w) and (w, s_2). Note that G has vertex-disjoint (s_1, t_1) and (s_2, t_2)-paths if and only if G' has an (s_1, t_2)-path containing w. We define the coloring c by setting $c(w) = 1$ and defining $c(v) = 2$ for all $v \in V(G') \setminus \{w\}$. Clearly, G' has a 2-colored (s_1, t_2)-path if and only if G' has an (s_1, t_2)-path containing w. This immediately implies NP-hardness. □

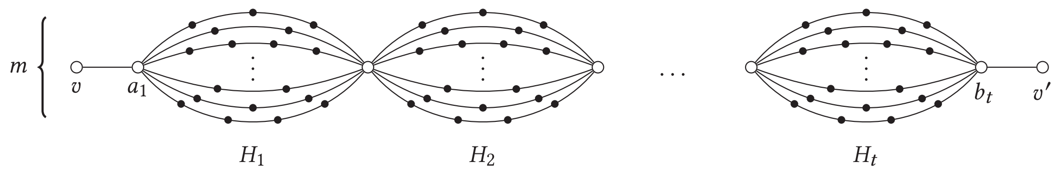


Fig. 4. Construction of the graph G.

Then, we show that Theorem 1.1 is optimal assuming the Set Cover Conjecture. In the Set Cover problem, we are given a universe U of n elements, a collection $\mathcal{S}$ of m subsets of U, and an integer t and we ask whether there is a collection $\mathcal{S}' \subseteq \mathcal{S}$ of size t such that for every $u \in U$, there is a set $S \in \mathcal{S}'$ such that $u \in S$.

Proposition 3.3. *If there is a $(2-\varepsilon)^k n^{O(1)}$ time algorithm for finding a k-colored path in a k-colored graph for some $\varepsilon > 0$, then there is a $(2-\varepsilon)^n (mn)^{O(1)}$ time algorithm for Set Cover, in a universe U of size n with a collection $\mathcal{S}$ of m subsets of U.*

Proof. Given an instance $(U, \mathcal{S}, t)$ of Set Cover, where $|U| = n$ and $\mathcal{S} = \{S_1, \dots, S_m\}$, we construct a graph G as follows. We first construct the graph H by considering two vertices a and b and adding m internally vertex-disjoint (a,b)-paths $P_{S_1}, \dots, P_{S_m}$, where for every $i \in [m]$, the vertices in P_{S_i} are bijectively mapped to the elements of S_i. We call a the *source* of H and b the *sink* of H. We finally construct a graph G that is obtained by considering t copies $H_1, \dots, H_t$ of H, for each $i \in [t-1]$, identifying the sink b_i of H_i with the source a_{i+1} of H_{i+1}, and adding two new vertices v and v' of degree one, adjacent to a_1 and b_t respectively. See Figure 4 for an illustration of the construction of graph G. Note that $t \le m$ and $|V(G)| = (mn)^{O(1)}$.

Assuming an ordering $u_1, \dots, u_n$ of U, for each $i \in [n]$, we assign color i to all vertices of G that correspond to u_i, color $n+1$ and $n+2$ to v and v', and color $n+3$ to all vertices in $V(G) \setminus \{v, v'\}$ that do not correspond to members of U. Observe that $(U, \mathcal{S}, t)$ is a yes-instance of Set Cover if and only if there is an $n+3$-colored path in G. Therefore, a $(2-\varepsilon)^k n^{O(1)}$ time algorithm for finding a k-colored path in a k-colored n-vertex graph implies the existence of a $(2-\varepsilon)^n (mn)^{O(1)}$ time algorithm for finding a set cover of size t in a universe U of size n with a collection $\mathcal{S}$ of m subsets of U. □

4 Randomized Algorithm for Colored (S,T)-Linkages

In this section, we prove the main result, i.e., Theorem 1.4. Recall that Theorem 1.1 is a special case of Theorem 1.4.

Let G be an n-vertex graph, p an integer, and $S, T \subseteq V(G)$. An (S,T)-linkage of order p is a set $\mathcal{P}$ of $p = |\mathcal{P}|$ vertex-disjoint paths between S and T. We denote by $V(\mathcal{P})$ the vertices in the paths of $\mathcal{P}$. The length of an (S,T)-linkage is the total number $|V(\mathcal{P})|$ of vertices in the paths. Let $c : V(G) \to [n]$ an arbitrary coloring of G, and $\mathtt{we} : V(G) \to \mathbb{Z}_{\geq 1}$ a weight function. For positive integers k and w, we say that an (S,T)-linkage $\mathcal{P}$ is (k,w)-colored if there exists a set $X \subseteq V(\mathcal{P})$ with $|X| = k$, all vertices of X have different colors, and $\mathtt{we}(X) = \sum_{v \in X} \mathtt{we}(v) = w$. We give a $2^{p+k} n^{O(1)} w$ time algorithm for the problem of finding a minimum length (k,w)-colored (S,T)-linkage of order p (Theorem 1.4).

We will assume that $|S| = |T| = p$, and S and T are disjoint, as the general case can be reduced to this case by the following reduction: We add p vertices $s_1, \dots, s_p$ with $N(s_i) = S$ and p vertices $t_1, \dots, t_p$ with $N(t_i) = T$, all with the same new color and weight equal to $k \cdot \max_{v \in V(G)} \mathtt{we}(v) + 1$. Then, we can set $S = \{s_1, \dots, s_p\}$ and $T = \{t_1, \dots, t_p\}$, and solve the problem with k increased by one and w increased by $k \cdot \max_{v \in V(G)} \mathtt{we}(v) + 1$. Because we can assume that the original weights

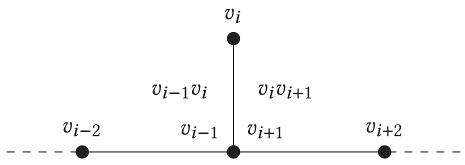


Fig. 5. An example of a labeled walk W with a digon i.

are at most $w + 1$, this increases the target weight w by a factor $O(k)$, and therefore does not increase the time complexity of the algorithm.

4.1 Labeled Walks and Walkages

In this section, we define labeled walks and labeled walkages.

Labeled walks. Let ℓ be an integer. A *walk of length* ℓ in G is a sequence of vertices $v_1, \dots, v_\ell$ of G, where $v_iv_{i+1} \in E(G)$ for all $1 \leq i < \ell$. A *labeled walk of length* ℓ is a pair of sequences $W = ((v_1, v_2, \dots, v_\ell), (r_1, r_2, \dots, r_\ell))$, where $v_1, \dots, v_\ell$ is a walk of length ℓ, and $r_1, \dots, r_\ell$ is a sequence of integers from $[0, k]$, indicating a labeling. The interpretation of the labeling is that $r_i = 0$ indicates that the index i is unlabeled and $r_i \neq 0$ indicates that the index i is labeled with the label $r_i \in [k]$. A labeled walk is *injective* if each label from $[k]$ appears in it at most once. Most of the labeled walks that we treat in the algorithm have length at least one, but the definition allows also an empty labeled walk of length zero. The set of vertices *collected* by W, denoted by $R(W)$, is the set of all vertices $v_i, i \in [\ell]$ such that $r_i \neq 0$, i.e., the set of vertices that occur at labeled indices. The set of edges of W is $E(W) = \{v_iv_{i+1} : 1 \leq i < \ell\}$. An index i in a labeled walk of length ℓ is a *digon* if $1 < i < \ell$ and $v_{i-1} = v_{i+1}$ (see Figure 5 for an illustration). An index i in a labeled walk is a *labeled digon* if it is a digon and $r_i \neq 0$.

Labeled walkages. A *labeled walkage of order p* is a tuple $\mathcal{W} = (W^1, \dots, W^p)$, where each for each $i \in [p]$, $W^i = ((v^i_1, \dots, v^i_{\ell_i}), (r^i_1, \dots, r^i_{\ell_i}))$ is a labeled walk of length $\ell_i \geq 1$. The *length* of $\mathcal{W}$ is $\sum_{i=1}^p \ell_i$. The *set of edges* of $\mathcal{W}$ is $E(\mathcal{W}) = \bigcup_{i=1}^p E(W^i)$. The set of vertices *collected* by $\mathcal{W}$ is $R(\mathcal{W}) = \bigcup_{i=1}^p R(W^i)$. The weight $\mathtt{we}(\mathcal{W})$ of $\mathcal{W}$ is the sum of the weights of the labeled vertices, i.e., $\mathtt{we}(\mathcal{W}) = \sum_{i=1}^p \sum_{j \in [\ell_i] : r^i_j \neq 0} \mathtt{we}(v^i_j)$. Note that the weight of a vertex can be counted more than once if the vertex corresponds to more than once labeled index. A labeled walkage is *injective* if each label from $[k]$ appears in it at most once, and *bijective* if each label from $[k]$ appears in it exactly once. Note that every labeled walk in an injective labeled walkage is injective.

The set of *ending* vertices of a labeled walkage $\mathcal{W}$ of order p is $\mathcal{T}(\mathcal{W}) = \{v^i_{\ell_i} : i \in [p]\}$. The tuple of *starting* vertices of $\mathcal{W}$ is $\mathtt{start}(\mathcal{W}) = (v^1_1, \dots, v^p_1)$. Let $<$ be a total order on $V(G)$. A labeled walkage is *ordered* if $\mathtt{start}(\mathcal{W})$ is ordered according to $<$, i.e., $v^i_1 < v^{i+1}_1$ holds for all $1 \leq i < p$. The asymmetry that the starting vertices are an ordered tuple while the ending vertices are an unordered set is essential for our algorithm. A *labeled linkage* is a labeled walkage where every vertex occurs at most once, i.e., the walks are vertex-disjoint paths.

We also define *semiproper* and *proper* labeled walkages. The intuition here is that, in Section 4.2, we define a polynomial over semiproper walkages (see also Definition 4.1). Then, walkages that are semiproper but not proper will be handled by using standard techniques and therefore we can focus on proper walkages. Dealing with proper walkages will be the most technical part of the proof. A labeled walkage is *semiproper* if it is injective, no walk in it contains labeled digons, and the ending vertices of the walkage are distinct, i.e., $v^i_{\ell_i} \neq v^j_{\ell_j}$ for $i \neq j$. A labeled walkage $\mathcal{W}$ is *proper* if it is semiproper and all of its labeled indices correspond to vertices of different colors, i.e.,

$|c(R(\mathcal{W}))| = |\{(i,j) : r^i_j \neq 0\}|$. Note that being proper implies that no vertex is labeled twice and note that if $\mathcal{W}$ is bijective and proper then $|c(R(\mathcal{W}))| = k$.

4.2 Algorithm

We assume that there is a total order $<$ on $V(G)$, and for a set $S \subseteq V(G)$ we denote by $\mathtt{ordv}(S)$ the tuple containing the elements of S ordered according to $<$. Note that G contains a (k,w)-colored (S,T)-linkage of order p and length ℓ if and only if there is a bijective proper ordered labeled linkage $\mathcal{W}$ with order p, length ℓ, weight $\mathtt{we}(\mathcal{W}) = w$, tuple of starting vertices $\mathtt{start}(\mathcal{W}) = \mathtt{ordv}(S)$, and set of ending vertices $\mathcal{T}(\mathcal{W}) = T$. We define a family of labeled walkages that includes all such labeled linkages but relaxes the condition of being a linkage to walkage and the condition of being proper to semiproper.

For each integer ℓ, we define a family of labeled walkages C_ℓ of length ℓ.

Definition 4.1 (Family C_ℓ). Let ℓ a positive integer. The family C_ℓ consists of the bijective semiproper ordered labeled walkages $\mathcal{W}$ with order p, length ℓ, weight $\mathtt{we}(\mathcal{W}) = w$, tuple of starting vertices $\mathtt{start}(\mathcal{W}) = \mathtt{ordv}(S)$, and set of ending vertices $\mathcal{T}(\mathcal{W}) = T$.

Definition of the polynomial. Let $q = 2^{3+\lceil \log_2 n \rceil}$ and keep in mind that GF(q) is a finite field of characteristic 2 and order $q \geq 8n$. Next, we define a polynomial over GF(q) that will be evaluated for a uniformly random assignment of values to the variables over GF(q) by our algorithm. For each edge $uv \in E(G)$ we associate a variable $f_{\mathrm{E}}(uv)$, for each vertex $v \in V(G)$ we associate a variable $f_{\mathrm{V}}(v)$, and for each color-label-pair $(x,y) \in [n] \times [k]$ we associate a variable $f_{\mathrm{C}}(x,y)$. In particular, the set of variables of the polynomial is $\bigcup_{uv \in E(G)} f_{\mathrm{E}}(uv) \cup \bigcup_{v \in V(G)} f_{\mathrm{V}}(v) \cup \bigcup_{(x,y) \in [n]\times[k]} f_{\mathrm{C}}(x,y)$. For a labeled walk $W = ((v_1, \ldots, v_\ell), (r_1, \ldots, r_\ell))$, we associate the monomial

$$f(W) = \prod_{i=1}^{\ell-1} f_{\mathrm{E}}(v_i v_{i+1}) \cdot \prod_{i \in [\ell]\colon r_i \neq 0} f_{\mathrm{V}}(v_i) \cdot f_{\mathrm{C}}(c(v_i), r_i).$$

For a labeled walkage $\mathcal{W} = (W^1, \ldots, W^p)$, we associate the monomial

$$f(\mathcal{W}) = \prod_{i=1}^{p} f(W^i).$$

For a family $\mathcal{F}$ of labeled walkages we associate the polynomial

$$f(\mathcal{F}) = \sum_{\mathcal{W} \in \mathcal{F}} f(\mathcal{W}).$$

Because the walkages in C_ℓ are bijective, every monomial in the polynomial $f(C_\ell)$ has degree $\ell - p + 2k$, being a product of $\ell - p$ variables corresponding to the edges of the walkage, k variables corresponding to the labeled vertices, and k variables corresponding to the color-label-pairs.

Algorithm for finding a (k,w)-colored (S,T)-linkage. Our algorithm for finding a (k,w)-colored (S,T)-linkage of order p works as follows. Starting with $\ell = p$, we evaluate the polynomial $f(C_\ell)$ for a uniformly random assignment of values to the variables over GF(q), for increasing values of ℓ. If the evaluation of $f(C_\ell)$ is not zero, we return that G contains a (k,w)-colored (S,T)-linkage of order p, and moreover that a shortest (k,w)-colored (S,T)-linkage of order p has length ℓ. Otherwise, we continue increasing ℓ until $\ell = n+1$ in which case we return that G does not contain a (k,w)-colored (S,T)-linkage of order p.

For the proof of correctness of the algorithm, in Section 4.3 we show that with probability at least $1/2$ this algorithm returns the length of a shortest (k,w)-colored (S,T)-linkage of order p, and never returns a length shorter than a shortest (k,w)-colored (S,T)-linkage of order p.

Proof of time complexity of the algorithm. Next we prove the time complexity of the algorithm. The evaluation of the polynomial is done using dynamic programming. This is a standard application of dynamic programming over walks while keeping track of the set of labels used so far, the weight of the labeled vertices, and the set of ending vertices used. We prove that it can be performed in time $2^{p+k}n^{O(1)}w$.

LEMMA 4.2. *Let S, T be disjoint subsets of $V(G)$ of size $|S| = |T| = p$, $c : V(G) \longrightarrow [n]$ a coloring of G, $\mathtt{we} : V(G) \longrightarrow \mathbb{Z}_{\geq 1}$ a weight function, $\ell \leq n$ an integer, k, w integers, and $q = 2^{3+\lceil \log_2 n \rceil}$. Given values in GF(q) for all of the variables $f_{\mathrm{E}}(uv)$ for $uv \in E(G)$, $f_{\mathrm{V}}(v)$ for $v \in V(G)$, and $f_{\mathrm{C}}(x, y)$ for $(x, y) \in [n] \times [k]$, the value of the polynomial $f(C_\ell)$ can be computed in time $2^{p+k}n^{O(1)}w$.*

PROOF. Let us denote by $\hat{f}_{\mathrm{E}}(uv)$, $\hat{f}_{\mathrm{V}}(v)$, and $\hat{f}_{\mathrm{C}}(x, y)$ these given values, and by extension for a walkage $\mathcal{W}$ denote by $\hat{f}(\mathcal{W})$ the value associated to the monomial $f(\mathcal{W})$ and for a family of walkages $\mathcal{F}$ denote by $\hat{f}(\mathcal{F})$ the value associated to the polynomial $f(\mathcal{F})$. The task is to compute $\hat{f}(C_\ell)$.

Informally, we will compute $\hat{f}(C_\ell)$ by dynamic programming over partial walkages, growing the walkages one labeled walk at a time in the order specified by $\mathtt{ordv}(S)$.

Denote $\mathtt{ordv}(S) = (s_1, s_2, \ldots, s_p)$ and for any $t \in [p]$ denote by $\mathtt{pre}_t(S)$ the length-t prefix of $\mathtt{ordv}(S)$. For every integer $t \in [p]$, integer $l \in [\ell]$, set $L \subseteq [k]$ of labels, set $T' \subseteq T$ of ending vertices, weight $w' \in [0, w]$, vertices $x, y \in V(G)$, and integer $o \in \{0, 1\}$, we define

$$D(t, l, L, T', w', x, y, o) = \hat{f}(\mathcal{F}(t, l, L, T', w', x, y, o)),$$

where we define $\mathcal{F}(t, l, L, T', w', x, y, o)$ to be the family of labeled walkages $\mathcal{W} = (W^1, \ldots, W^t)$, where for each $i \in [t]$, we have that $W^i = ((v^i_1, \ldots, v^i_{\ell_i}), (r^i_1, \ldots, r^i_{\ell_i}))$, and that satisfy the following properties:

(1) Each labeled walk W^i in $\mathcal{W}$ has length at least 2 and does not contain labeled digons,
(2) $\mathcal{W}$ has order t and ordered tuple of starting vertices $\mathtt{start}(\mathcal{W}) = \mathtt{pre}_t(S)$,
(3) $\mathcal{W}$ has length l,
(4) $\mathcal{W}$ is injective and the set of used labels is L,
(5) the set of ending vertices of all but the last walk in $\mathcal{W}$ is $\mathcal{T}((W^1, \ldots, W^{t-1})) = T'$,
(6) $\mathcal{W}$ has weight $\mathtt{we}(\mathcal{W}) = w'$,
(7) the last vertex of the last walk in $\mathcal{W}$ is $v^t_{\ell_t} = x$,
(8) the second last vertex of the last walk in $\mathcal{W}$ is $v^t_{\ell_t - 1} = y$, and
(9) if $o = 0$, then $r_{t,\ell_t} = 0$, otherwise $r_{t,\ell_t} \neq 0$.

In other words, t specifies the number of walks, l specifies the total length, L specifies the used labels, T' specifies the used ending vertices, w' specifies the weight, x specifies the last vertex of the last walk, y specifies the second last vertex of the last walk, and o specifies whether the last vertex of the last walk is labeled. Note that it can be without loss of generality assumed that each walk has length at least 2 because S and T are disjoint.

Then, we define also a shorthand that for $t \in [p]$, $l \in [\ell]$, $L \subseteq [k]$, $T' \subseteq T$, and $w' \in [0, w]$,

$$D(t, l, L, T', w') = \sum_{x \in T'} \sum_{y \in N(x)} \sum_{o \in \{0,1\}} D(t, l, L, T' \setminus \{x\}, w', x, y, o),$$

which intuitively denotes the polynomial corresponding to a "completed" walkage of t walks with length l, used labels L, used ending vertices T', and weight w'.

Now it holds that

$$\hat{f}(C_\ell) = D(p, \ell, [k], T, w),$$

and therefore computing $\hat{f}(C_\ell)$ can be done by computing all of the values $D(t, l, L, T', w', x, y, o)$ by dynamic programming.

Next, we specify this computation by dynamic programming. All values that we do not specify here will be set to zero. First, to initialize, we define a special value $D(0, 0, \emptyset, \emptyset, 0) = 1$ corresponding to a family of walkages containing one empty walkage.

Next, we describe computing the states where $o = 0$, i.e., the last vertex is not labeled, for all $t \in [p]$, $l \in [\ell]$, $L \subseteq [k]$, $T' \subseteq T$, $w' \in [0, w]$, $x \in V(G)$, and $y \in N(x)$ (if x and y are not adjacent, we set the value to zero), assuming that all the states with smaller l have already been computed. There are four cases, corresponding to the four lines of Equation (1), which we state shortly. In the first case the walk W_t has length at least three, its second last vertex y is not labeled, and we are extending the walkage by adding one not labeled vertex x to W_t. Second case is the same, but the second last vertex y is labeled and thus we have to ensure to not create a labeled digon. Third case is the case that we are extending the walkage by adding one more labeled walk, consisting of two vertices y, x, where $y = s_t$, neither of them labeled. Fourth case is like the third, but the first vertex $y = s_t$ of the new walk is labeled. Recall the notation that $[y = s_t] = 1$ if $y = s_t$ holds, and 0 otherwise.

$$\begin{aligned}
D(t, l, L, T', w', x, y, 0) &= f_{\mathrm{E}}(xy) \\
&\cdot \left(\sum_{z \in V(G)} D(t, l-1, L, T', w', y, z, 0) \right. \\
&+ \sum_{z \in V(G) \setminus \{x\}} D(t, l-1, L, T', w', y, z, 1) \\
&+ [y = s_t] \cdot D(t-1, l-2, L, T', w') \\
&\left. + [y = s_t] \cdot \sum_{r \in L} f_{\mathrm{V}}(y) \cdot f_{\mathrm{C}}(c(y), r) \cdot D(t-1, l-2, L \setminus \{r\}, T', w' - \mathtt{we}(y)) \right).
\end{aligned} \tag{1}$$

Then, we describe computing the states where $o = 1$, i.e., the last vertex is labeled, for all $t \in [p]$, $l \in [\ell]$, $L \subseteq [k]$, $T' \subseteq T$, $w' \in [0, w]$, $x \in V(G)$, and $y \in N(x)$, assuming that all of the states with smaller l have already been computed. There are again four cases, analogously to Equation (1).

$$\begin{aligned}
D(t, l, L, T', w', x, y, 1) &= \sum_{r \in L} f_{\mathrm{V}}(x) \cdot f_{\mathrm{C}}(c(x), r) \cdot f_{\mathrm{E}}(xy) \\
&\cdot \left(\sum_{z \in V(G)} D(t, l-1, L \setminus \{r\}, T', w' - \mathtt{we}(x), y, z, 0) \right. \\
&+ \sum_{z \in V(G) \setminus \{x\}} D(t, l-1, L \setminus \{r\}, T', w' - \mathtt{we}(x), y, z, 1) \\
&+ [y = s_t] \cdot D(t-1, l-2, L \setminus \{r\}, T', w' - \mathtt{we}(x)) \\
&\left. + [y = s_t] \cdot \sum_{r' \in L \setminus \{r\}} f_{\mathrm{V}}(y) \cdot f_{\mathrm{C}}(c(y), r') \cdot D(t-1, l-2, L \setminus \{r, r'\}, T', w' - \mathtt{we}(x) - \mathtt{we}(y)) \right).
\end{aligned} \tag{2}$$

This completes the description of the dynamic programming, showing that each of the states $D(t, l, L, T', w', x, y, o)$ can be computed in $n^{O(1)}$ time given the values of the states with smaller l. As there are $p \cdot \ell \cdot 2^k \cdot 2^p \cdot (w+1) \cdot n \cdot n \cdot 2 = O(p 2^{p+k} n^3 w)$ states, the algorithm works in time $2^{p+k} n^{O(1)} w$. □

As the algorithm can be implemented by $O(n)$ applications of Lemma 4.2, the algorithm has time complexity $2^{p+k}n^{O(1)}w$. Recovering the solution can be done by a factor of $O(n^2)$ more applications.

4.3 Correctness

To prove the correctness of the algorithm, we show that

(a) the polynomial $f(C_\ell)$ is not the zero polynomial if G contains a (k, w)-colored (S, T)-linkage of order p and length ℓ and
(b) the polynomial $f(C_\ell)$ is the zero polynomial if the graph does not contain a (k, w)-colored (S, T)-linkage of order p and length $\leq \ell$.

As $f(C_\ell)$ has degree $\ell - p + 2k \leq 3n \leq q/2$, (a) implies, by applying Lemma 3.1 with $p = f(C_\ell)$ and $\{x_1, \ldots, x_r\} = \bigcup_{uv \in E(G)} f_E(uv) \cup \bigcup_{v \in V(G)} f_V(v) \cup \bigcup_{(x,y) \in [n] \times [k]} f_C(x, y)$, that if G contains a (k, w)-colored (S, T)-linkage of order p and length ℓ, then evaluating $f(C_\ell)$ for a uniformly random assignment of values to the variables over GF(q) has probability at least $1/2$ to not be zero. From (b) it follows that if G does not contain a (k, w)-colored (S, T)-linkage of order p and length $\leq \ell$, then the evaluation of $f(C_\ell)$ for any assignment of values to the variables is guaranteed to be zero. This establishes that the algorithm is correct with probability at least $1/2$, with one-sided error.

The part (a) is relatively easy to prove (Lemma 4.3). To prove (b), we first show that the monomials in $f(C_\ell)$ corresponding to non-proper labeled walkages cancel out (Lemma 4.4). This argument is based on the now-standard technique of bijective labeling based cancellation introduced in [2]. The remaining part of the proof of (b) is much more complicated and is the main technical challenge. It is based on the technical Lemma 4.6, whose proof is postponed to Section 4.4.

We start with (a).

Lemma 4.3. *If G has a (k, w)-colored (S, T)-linkage of order p and length ℓ, then $f(C_\ell)$ is not the zero polynomial.*

Proof. Consider a (k, w)-colored (S, T)-linkage $\mathcal{P}$ of order p and length ℓ. Let $X \subseteq V(\mathcal{P})$ be the set of vertices with $|X| = k$, different colors, and weight $\texttt{we}(X) = w$. We can turn $\mathcal{P}$ into a proper labeled linkage $\mathcal{W}$ of order p, length ℓ, weight w, where $\texttt{start}(\mathcal{W}) = \texttt{ordv}(S)$ and $\mathcal{T}(\mathcal{W}) = T$, by ordering the paths based on their starting vertices and assigning the labels $[k]$ arbitrarily to the vertices X when $\mathcal{W}$ intersects X.

Therefore, $\mathcal{W} \in C_\ell$, so it remains to prove that $\mathcal{W}$ is the only labeled walkage in C_ℓ that corresponds to the monomial $f(\mathcal{W})$, which then implies that the monomial $f(\mathcal{W})$ occurs in the polynomial $f(C_\ell)$ with coefficient 1, implying that $f(C_\ell)$ is not the zero polynomial.

Notice that from $f(\mathcal{W})$, from the edge variables f_E we can recover the edges $E(\mathcal{W})$ of $\mathcal{W}$, from the vertex variables f_V we can recover the labeled vertices X, and because vertices in X have different colors, from the color-label pair variables f_C we can recover how the labels correspond to the labeled vertices. Therefore as the ordering of the paths is fixed by $\texttt{ordv}(S)$ and every vertex appears in $\mathcal{W}$ at most once, we have that $\mathcal{W}$ is the unique element of C_ℓ that corresponds to the monomial $f(\mathcal{W})$. □

Then, we deal with non-proper walkages in C_ℓ. Let $C_\ell^* \subseteq C_\ell$ denote the family of proper labeled walkages in C_ℓ, i.e., the labeled walkages in C_ℓ where all labeled indices have vertices of different colors.

Lemma 4.4. *It holds that $f(C_\ell^*) = f(C_\ell)$.*

Proof. We will show that there is a function $\phi : C_\ell \setminus C_\ell^* \to C_\ell \setminus C_\ell^*$ that is an f-invariant fixed-point-free involution, i.e., for all $\mathcal{W} \in C_\ell \setminus C_\ell^*$ it holds that (1) $f(\phi(\mathcal{W})) = f(\mathcal{W})$, (2) $\phi(\mathcal{W}) \neq \mathcal{W}$, and (3) $\phi(\phi(\mathcal{W})) = \mathcal{W}$. This implies that the set $C_\ell \setminus C_\ell^*$ can be partitioned into pairs $\{\mathcal{W}, \phi(\mathcal{W})\}$ with $f(\mathcal{W}) = f(\phi(\mathcal{W}))$, and therefore every monomial corresponding to a labeled walkage in $C_\ell \setminus C_\ell^*$ occurs in $f(C_\ell)$ an even number of times, and therefore they cancel out because f is over a field of characteristic 2.

The function ϕ is defined as follows. Let $\mathcal{W} = (W^1, \dots, W^p)$ be a labeled walkage in $C_\ell \setminus C_\ell^*$, where $W^i = ((v_1^i, \dots, v_{\ell_i}^i), (r_1^i, \dots, r_{\ell_i}^i))$. Because $\mathcal{W}$ is semiproper but not proper, there exists two different labeled indices that have a vertex of the same color, i.e., pairs (i, a) and (j, b) with $i, j \in [p]$, $a \in [\ell_i]$, $b \in [\ell_j]$, $(i, a) \neq (j, b)$, $c(v_a^i) = c(v_b^j)$, $r_a^i \neq 0$, and $r_b^j \neq 0$. Let $(i, a), (j, b)$ be the lexicographically smallest such pair. We set $\phi(\mathcal{W})$ to be the labeled walkage obtained from $\mathcal{W}$ after swapping r_a^i with r_b^j.

First, we observe that $\phi(\mathcal{W}) \in C_\ell$. Indeed, it cannot make a bijective walkage into non-bijective, and as it does not change the sequence of vertices of $\mathcal{W}$ or which indices are labeled, it cannot make a semiproper walk into non-semiproper, or change the order, the length, the weight, the tuple of starting vertices, or the set of ending vertices. Also $\phi(\mathcal{W})$ is not proper, i.e., $\phi(\mathcal{W}) \in C_\ell \setminus C_\ell^*$, because the vertices v_a^i and v_b^j are still labeled and have the same color.

To see why $f(\phi(\mathcal{W})) = f(\mathcal{W})$, note that, since ϕ does not change the vertices, it also does not change the edge variables f_{E} of the monomial, it does not change which vertices are labeled so it does not change the vertex variables f_{V} of the monomial, and because the vertices v_a^i and v_b^j have the same color the color-label-pair variables f_{C} of the monomial are also not changed.

Also, we have that $\phi(\mathcal{W}) \neq \mathcal{W}$, since the fact that $\mathcal{W}$ is bijective implies that $r_a^i \neq r_b^j$. Also, $\phi(\phi(\mathcal{W})) = \mathcal{W}$ because the swapping does not change which indices are labeled, and therefore does not change the lexicographically smallest pair of labeled indices with the same colors. □

As a result of Lemma 4.4, we can work with $f(C_\ell^*)$ instead of $f(C_\ell)$.

The most complicated part of the correctness proof will be to show part (b), that is, if there is no (k, w)-colored (S, T)-linkage of order p and length at most ℓ, then $f(C_\ell^*)$ (and, thus by Lemma 4.4, $f(C_\ell)$) is the zero polynomial. Most of this proof will be presented in Section 4.4, but we introduce here the statement of the lemma that we will prove in Section 4.4. For this, we define *barren labeled walkages.*

Definition 4.5 (Barren labeled walkage). A labeled walkage $\mathcal{W}$ of length ℓ is *barren* if there exists no labeled linkage $\mathcal{W}'$ with starting vertices $\texttt{start}(\mathcal{W}') = \texttt{start}(\mathcal{W})$, set of ending vertices $\mathcal{T}(\mathcal{W}') = \mathcal{T}(\mathcal{W})$, set of collected vertices $R(\mathcal{W}') = R(\mathcal{W})$, length $\leq \ell$ and edges $E(\mathcal{W}') \subseteq E(\mathcal{W})$.

In other words, a labeled walkage $\mathcal{W}$ of length ℓ is barren if its edges form a subgraph of G where no labeled linkage$\mathcal{W}'$ of length at most ℓ can have the same sets of starting vertices, ending vertices, and collected vertices as $\mathcal{W}$. Intuitively, this means that the labeled walkage $\mathcal{W}$ cannot be "untangled" to give a corresponding labeled linkage. In particular, observe that because the "untangling" preserves the set of collected vertices, i.e., $R(\mathcal{W}') = R(\mathcal{W})$, if no (k, w)-colored (S, T)-linkages of order p and length at most ℓ exists, then all labeled walkages in C_ℓ^* are barren.

Next, we state the main technical lemma for establishing the correctness of our algorithm. Section 4.4 is devoted to its proof.

Lemma 4.6. *Let G be a graph and let $\mathcal{B}$ the set of all proper barren labeled walkages in G. There exists a function $\phi : \mathcal{B} \to \mathcal{B}$ so that for all $\mathcal{W} \in \mathcal{B}$, the function ϕ satisfies that*

(1) $\phi(\phi(\mathcal{W})) = \mathcal{W}$ (ϕ is involution),
(2) $\phi(\mathcal{W}) \neq \mathcal{W}$ (ϕ is fixed-point-free),
(3) $f(\phi(\mathcal{W})) = f(\mathcal{W})$ (ϕ preserves the monomial),

(4) $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$ *(ϕ preserves the set of ending vertices), and*
(5) $\texttt{start}(\phi(\mathcal{W})) = \texttt{start}(\mathcal{W})$ *(ϕ preserves the ordered tuple of starting vertices).*

The main reason for defining the function ϕ for all proper barren labeled walkages instead of just barren walkages in C_ℓ^* is that ϕ will be defined recursively, and in the recursion we will anyway need to handle all proper barren labeled walkages.

Now, the proof of (b) is an easy consequence of Lemma 4.6.

LEMMA 4.7. *If G has no (k, w)-colored (S, T)-linkage of order p and length $\leq \ell$, then $f(C_\ell^*)$ is the zero polynomial.*

PROOF. First, because G has no (k, w)-colored (S, T)-linkage of order p and length $\leq \ell$, all labeled walkages in C_ℓ^* are barren, i.e., $C_\ell^* \subseteq \mathcal{B}$.

We show that if $\mathcal{W} \in C_\ell^*$, then $\phi(\mathcal{W}) \in C_\ell^*$. By definition, $\phi(\mathcal{W})$ is proper. By (3), ϕ preserves the set of labeled vertices and moreover because the labeled vertices have different colors it preserves also the label-vertex mapping, and therefore $\phi(\mathcal{W})$ is bijective and has weight w. By (4) and (5), ϕ preserves the set of ending vertices and the ordered tuple of starting vertices. By (3), ϕ also preserves the length ℓ, as the order of $\mathcal{W}$ is preserved by (5). Therefore the restriction $\phi \restriction_{C_\ell^*}$ is a function $\phi \restriction_{C_\ell^*} \colon C_\ell^* \to C_\ell^*$.

Then, by (1)–(3), $\phi \restriction_{C_\ell^*}$ is an f-invariant fixed-point-free involution on C_ℓ^*, implying that the set C_ℓ^* can be partitioned into pairs $\{\mathcal{W}, \phi(\mathcal{W})\}$ with $f(\mathcal{W}) = f(\phi(\mathcal{W}))$, and therefore for every monomial $f(\mathcal{W})$, there is an even number of labeled walkages $\mathcal{W} \in C_\ell^*$ corresponding to it, and therefore because $f(C_\ell^*)$ is over a field of characteristic 2, it is the zero polynomial. □

4.4 Proof of Lemma 4.6

In this section, we prove Lemma 4.6 by explicitly defining the function ϕ and then showing that it has all of the required properties.

In order to define ϕ we first introduce some notation for manipulating labeled walks and labeled walkages. Let $W = ((v_1, \dots, v_\ell), (r_1, \dots, r_\ell))$ be a labeled walk. For indices a, b with $1 \leq a \leq b \leq \ell$, we denote by $W[a, b]$ the labeled subwalk between a and b, inclusive, i.e., the labeled walk $W[a, b] = ((v_a, \dots, v_b), (r_a, \dots, r_b))$. If $a > b$, then $W[a, b]$ denotes an empty labeled walk.

The involution ϕ will use three types of operations: reversing a subwalk, swapping a label from one occurrence of a vertex to another occurrence of it (possibly in a different walk), and swapping suffixes of two walks.

The subwalk reversal operation is defined as follows. Let W be a labeled walk of length ℓ and a, b indices with $1 \leq a \leq b \leq \ell$. The walk obtained from W by reversing the subwalk between a and b, inclusive, including the labels, is denoted by $W\overleftarrow{[a, b]}$. For example, if $W = ((v_1, v_2, v_3, v_4), (0, 1, 0, 2))$, then $W\overleftarrow{[2, 3]} = ((v_1, v_3, v_2, v_4), (0, 0, 1, 2))$. A labeled walk W is a *palindrome* if $W = W\overleftarrow{[1, \ell]}$ holds, i.e., the labeled walk is the same in reverse. Note that $W\overleftarrow{[a, b]} = W$ holds if and only if $W[a, b]$ is a palindrome. Note also that because of the absence of self-loops, a subwalk $W[a, b]$ can be a palindrome only if its length is odd or if it is the empty walk. We will use the following lemma about palindromic subwalks of labeled walks, and in particular the reason to forbid labeled digons is to make this lemma true. Recall that any labeled walk in a proper labeled walkage is injective and does not contain labeled digons. Recall also that $R(W[a + 1, b - 1]) = \emptyset$ if and only if W has no labels in the subwalk $W[a + 1, b - 1]$.

LEMMA 4.8. *Let $W = ((v_1, \dots, v_\ell), (r_1, \dots, r_\ell))$ be an injective labeled walk of length ℓ that does not contain labeled digons, and let $a, b \in [\ell]$. If $v_a = v_b$ and $W[a + 1, b - 1]$ is a palindrome, then $R(W[a + 1, b - 1]) = \emptyset$.*

$$\begin{array}{cc}
\left((v^1_1,\dots,v^1_{\ell_1}),(r^1_1,\dots,r^1_{\ell_1})\right) & \left((v^1_1,\dots,v^1_{\ell_1}),(r^1_1,\dots,r^1_{\ell_1})\right) \\
\vdots & \vdots \\
\left((v^i_1,\dots,v^i_a,\dots,v^i_{\ell_i}),(r^i_1,\dots,r^i_a,\dots,r^i_{\ell_i})\right) & \left((v^i_1,\dots,v^i_a,\dots,v^i_{\ell_i}),(r^i_1,\dots,r^j_b,\dots,r^i_{\ell_i})\right) \\
\left((v^j_1,\dots,v^j_b,\dots,v^j_{\ell_j}),(r^j_1,\dots,r^j_b,\dots,r^j_{\ell_j})\right) & \left((v^j_1,\dots,v^j_b,\dots,v^j_{\ell_j}),(r^j_1,\dots,r^i_a,\dots,r^j_{\ell_j})\right) \\
\vdots & \vdots \\
\left((v^p_1,\dots,v^p_{\ell_p}),(r^p_1,\dots,r^p_{\ell_p})\right) & \left((v^p_1,\dots,v^p_{\ell_p}),(r^p_1,\dots,r^p_{\ell_p})\right) \\
 & \\
\mathcal{W} & \mathcal{W}\frown^{i,j}_{a,b}
\end{array}$$

Fig. 6. An illustration of the label swap operation. On the left: a labeled walkage $\mathcal{W} = (W^1,\dots,W^p)$, and pairs (i,a), (j,b) with $i,j \in [p]$, $a \in [\ell_i]$, $b \in [\ell_j]$, $v^i_a = v^j_b$, and exactly one of r^i_a and r^j_b equal to zero. Note that we allow $i = j$. On the right: the labeled walkage $\mathcal{W}\frown^{i,j}_{a,b}$.

$$\begin{array}{cc}
\left((v^1_1,\dots,v^1_{\ell_1}),(r^1_1,\dots,r^1_{\ell_1})\right) & \left((v^1_1,\dots,v^1_{\ell_1}),(r^1_1,\dots,r^1_{\ell_1})\right) \\
\vdots & \vdots \\
\left((v^i_1,\dots,v^i_a,\dots,v^i_{\ell_i}),(r^i_1,\dots,r^i_a,\dots,r^i_{\ell_i})\right) & \left((v^i_1,\dots,v^j_b,\dots,v^j_{\ell_j}),(r^i_1,\dots,r^j_b,\dots,r^j_{\ell_j})\right) \\
\left((v^j_1,\dots,v^j_b,\dots,v^j_{\ell_j}),(r^j_1,\dots,r^j_b,\dots,r^j_{\ell_j})\right) & \left((v^j_1,\dots,v^i_a,\dots,v^i_{\ell_i}),(r^j_1,\dots,r^i_a,\dots,r^i_{\ell_i})\right) \\
\vdots & \vdots \\
\left((v^p_1,\dots,v^p_{\ell_p}),(r^p_1,\dots,r^p_{\ell_p})\right) & \left((v^p_1,\dots,v^p_{\ell_p}),(r^p_1,\dots,r^p_{\ell_p})\right) \\
 & \\
\mathcal{W} & \mathcal{W}\leftrightarrow^{i,j}_{a,b}
\end{array}$$

Fig. 7. An illustration of the suffix swap operation. On the left: a labeled walkage $\mathcal{W} = (W^1,\dots,W^p)$, and pairs (i,a), (j,b) with $i,j \in [p]$, $a \in [\ell_i+1]$, $b \in [\ell_j+1]$, and $i \neq j$. On the right: the labeled walkage $\mathcal{W}\leftrightarrow^{i,j}_{a,b}$.

Proof. First, because $W[a+1,b-1]$ is injective and palindrome, the only vertex of $W[a+1,b-1]$ that can be labeled (i.e., have $r_i \neq 0$) is the middle vertex. However, a label cannot occur at the middle vertex of a palindrome with more than one vertex because it would be a labeled digon. If $W[a+1,b-1]$ has exactly one vertex, then again this vertex cannot be labeled because $v_a = v_b$ and W does not contain labeled digons. □

The label swap operation is defined as follows. Let $\mathcal{W} = (W^1,\dots,W^p)$ be a labeled walkage of order p, where for each $i \in [p]$ the walkage W^i is denoted by $((v^i_1,\dots,v^i_{\ell_i}),(r^i_1,\dots,r^i_{\ell_i}))$. Let (i,a), (j,b) be pairs with $i,j \in [p]$, $a \in [\ell_i]$, $b \in [\ell_j]$, $v^i_a = v^j_b$, and exactly one of r^i_a and r^j_b equal to zero (i.e., one of them unlabeled and one labeled). The labeled walkage obtained from $\mathcal{W}$ by swapping r^i_a with r^j_b is denoted by $\mathcal{W}\frown^{i,j}_{a,b}$. Note that because $r^i_a \neq r^j_b$, it holds that $\mathcal{W}\frown^{i,j}_{a,b} \neq \mathcal{W}$. See Figure 6 for an illustration of the label swap operation.

The suffix swap operation is defined as follows. Let (i,a) and (j,b) be pairs with $i,j \in [p]$, $a \in [\ell_i+1]$, $b \in [\ell_j+1]$, and $i \neq j$. The labeled walkage obtained from $\mathcal{W}$ by swapping the suffix of W^i starting at index a with the suffix of W^j starting at index b, including the labels, is denoted by $\mathcal{W}\leftrightarrow^{i,j}_{a,b}$. Note that here we allow that $a = \ell_i + 1$ or $b = \ell_j + 1$, with the interpretation that this corresponds to the empty suffix. Clearly, if both $a = \ell_i + 1$ and $b = \ell_j + 1$, then this operation does not do anything, but otherwise if $\mathcal{W}$ is a proper labeled walkage, applying this operation will in fact always result in a different walkage because of the different ending vertices condition. See Figure 7 for an illustration of the suffix swap operation.

If W^1 and W^2 are labeled walks so that the last vertex of W^1 is adjacent to the first vertex of W^2, then $W^1 \circ W^2$ denotes the concatenation of W^1 and W^2. If $\mathcal{W} = (W^1, \dots, W^p)$ is a labeled walkage and W is a labeled walk, then $W \diamond \mathcal{W}$ denotes the labeled walkage $(W \circ W^1, \dots, W^p)$ and $W \sqcup \mathcal{W}$ denotes the labeled walkage $(W, W^1, \dots, W^p)$.

Next, we define the function ϕ of Lemma 4.6. We will provide some intuition about ϕ right after the definition, and Figures 8–16 demonstrate different cases of it. The definition of ϕ will be recursive, using induction on the length of the walkage.

Definition 4.9 (The function ϕ). Let $\mathcal{W} = (W^1, \dots, W^p)$ be a proper barren labeled walkage of order p. For each $i \in [p]$, denote $W^i = ((v^i_1, \dots, v^i_{\ell_i}), (r^i_1, \dots, r^i_{\ell_i}))$. The value $\phi(\mathcal{W})$ is defined, in some cases recursively, by selecting the first matching case from the following list:

(A) if the vertex v^1_1 occurs only once in $\mathcal{W}$:
  (1) if $\ell_1 \geq 2$, then $\phi(\mathcal{W}) = W^1[1,1] \diamond \phi(W^1[2,\ell_1], W^2, \dots, W^p)$.
  (2) otherwise (i.e., $\ell_1 = 1$), $\phi(\mathcal{W}) = W^1 \sqcup \phi(W^2, \dots, W^p)$.

(B) if the vertex v^1_1 occurs in at least three different walks W^i: There must be at least two different walks W^i that contain v^1_1 but do not contain it as labeled. Let i, j be the two smallest indices so that both W^i and W^j contain v^1_1 but do not contain it as labeled. Let a be the index of the first occurrence of v^1_1 in W^i and b be the index of the first occurrence of v^1_1 in W^j. Now, $\phi(\mathcal{W}) = \mathcal{W} \leftrightarrow^{i,j}_{a,b}$.

(C) if the vertex v^1_1 occurs only in the walk W^1: By the case (A), the vertex v^1_1 occurs multiple times in W^1. Let b be the index of the last occurrence of v^1_1 in W^1 and a be the index of the second last occurrence of v^1_1 in W^1. Note that $a = 1$ if v^1_1 occurs only twice in W^1, and note also that $1 \leq a \leq b - 2$.
  (1) if $r^1_1 = r^1_b = 0$:
    (a) if $W^1[2, b-1]$ is not a palindrome, then $\phi(\mathcal{W}) = (W^1\overleftarrow{[2, b-1]}, W^2, \dots, W^p)$.
    (b) otherwise, if $b < \ell_1$, then $\phi(\mathcal{W}) = W^1[1, b] \diamond \phi(W^1[b+1, \ell_1], W^2, \dots, W^p)$.
    (c) otherwise (i.e., $b = \ell_1$), $\phi(\mathcal{W}) = W^1 \sqcup \phi(W^2, \dots, W^p)$.
  (2) if the index b is not a digon in W^1, then $\phi(\mathcal{W}) = \mathcal{W} \frown^{1,1}_{1,b}$. *Note: If neither case (1) nor (2) applies, then $r^1_1 \neq 0$.*
  (3) if $W^1[2, a-1]$ is not a palindrome, then $\phi(\mathcal{W}) = (W^1\overleftarrow{[2, a-1]}, W^2, \dots, W^p)$. *Note: If $a = 1$, then $W^1[2, a-1]$ is the empty walk which is a palindrome.*
  (4) if $v^1_{a+1} = v^1_{b-1}$:
    (a) if $W^1[a+1, b-1]$ is not a palindrome, then $\phi(\mathcal{W}) = (W^1\overleftarrow{[a+1, b-1]}, W^2, \dots, W^p)$.
    (b) otherwise, $\phi(\mathcal{W}) = W^1[1, b] \diamond \phi(W^1[b+1, \ell_1], W^2, \dots, W^p)$. *Note: Here $W^1[b+1, \ell_1]$ cannot be an empty walk because by case (C.2) b is a digon in W^1.*
  X. otherwise, $\phi(\mathcal{W}) = W^1[1, a] \diamond \phi(W^1[a+1, \ell_1], W^2, \dots, W^p)$. *Note: The case C.X will form a "common case" with the case D.X.*

(D) if the vertex v^1_1 occurs in exactly two different walks: Let i be the index of the another walk W^i in which v^1_1 occurs and let b be the index of the first occurrence of v^1_1 in W^i.
  (1) if $r^1_1 = r^i_b = 0$, then $\phi(\mathcal{W}) = \mathcal{W} \leftrightarrow^{1,i}_{1,b}$.

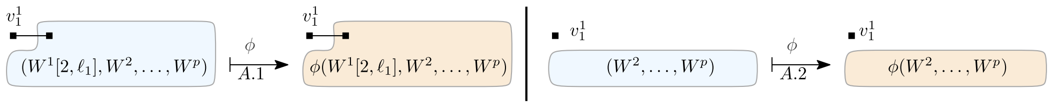

v_1^1
$(W^1[2,\ell_1], W^2, \ldots, W^p)$
ϕ
A.1
$\phi(W^1[2,\ell_1], W^2, \ldots, W^p)$
$(W^2, \ldots, W^p)$
A.2
$\phi(W^2, \ldots, W^p)$

Fig. 8. Examples of cases A.1 and A.2 of the definition of ϕ. The vertex v_1^1 can be either labeled or unlabeled.

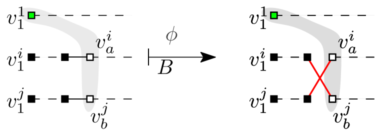


Fig. 9. Example of case B of the definition of ϕ. All vertices inside the gray bag are different occurrences of the same vertex v_1^1 of the graph. The white vertices v_a^i and v_b^j are unlabeled, the black vertices could be labeled or unlabeled, and the green vertex v_1^1 is labeled in this specific example.

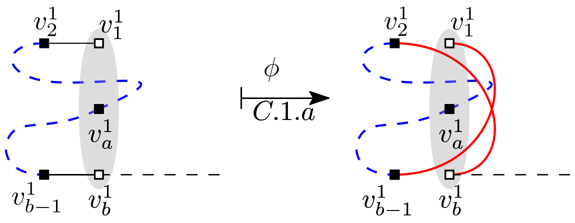


Fig. 10. Example of case C.1.a of the definition of ϕ. All vertices inside the gray bag are different occurrences of the same vertex v_1^1 of the graph. The vertices v_1^1 and v_b^1 are unlabeled and the black vertices v_2^1, v_a^1, and v_{b-1}^1 can be either labeled or unlabeled.

(2) if the index b is not a digon in W^i, then $\phi(\mathcal{W}) = \mathcal{W} \frown^{1,i}_{1,b}$. *Note: If neither case (1) nor (2) applies, then* $r_1^1 \neq 0$.

(3) if v_1^1 occurs at least twice in W^i, then let c be the index of its second occurrence and $\phi(\mathcal{W}) = \mathcal{W} \leftrightarrow^{1,i}_{2,c+1}$. *Note: It can happen that one of the suffixes in this case is empty. However, both of them cannot be empty at the same time because W^1 and W^i have different ending vertices because $\mathcal{W}$ is proper.*

Note: In the remaining cases, v_1^1 occurs exactly once in W^i, and this occurrence is a digon at index b.

Now, let a be the index of the last occurrence of v_1^1 in W^1 (if v_1^1 occurs only once in W^1, then $a = 1$).

(4) if $W^1[2, a-1]$ is not a palindrome, then $\phi(\mathcal{W}) = (\overleftarrow{W^1[2, a-1]}, W^2, \ldots, W^p)$. *Note: If $a = 1$, then $W^1[2, a-1]$ is the empty walk which is a palindrome.*

(5) if $a = \ell_1$, then $\phi(\mathcal{W}) = W^1 \sqcup \phi(W^2, \ldots, W^p)$.

(6) if $v_{a+1}^1 = v_{b+1}^i$, then $\phi(\mathcal{W}) = \mathcal{W} \leftrightarrow^{1,i}_{a+1,b+1}$. *Note: By case (5) it holds that $a < \ell_1$ and by case (2) it holds that $b < \ell_i$.*

X. otherwise, $\phi(\mathcal{W}) = W^1[1, a] \diamond \phi(W^1[a+1, \ell_1], W^2, \ldots, W^p)$. *Note: The case D.X will form a "common case" with the case C.X.*

Intuition for ϕ. Before laboriously proving that ϕ indeed is a function from proper barren labeled walkages to proper barren labeled walkages satisfying the required properties, let us give some rough outline of ideas behind it. First, the general idea is that if the walkage $\mathcal{W}$ goes to a certain

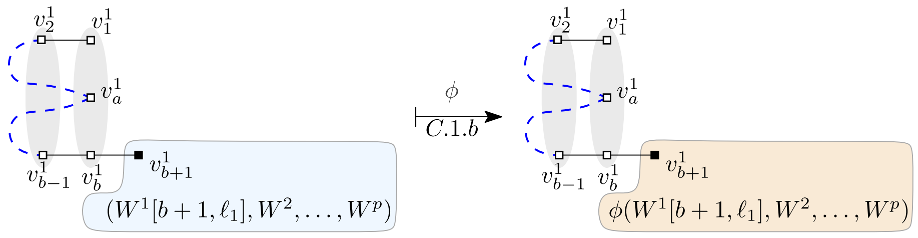

v_2^1
v_1^1
v_a^1
v_{b-1}^1
v_b^1
v_{b+1}^1
$(W^1[b+1,\ell_1], W^2, \dots, W^p)$
ϕ
$C.1.b$
$\phi(W^1[b+1,\ell_1], W^2, \dots, W^p)$

Fig. 11. Example of case C.1.b of the definition of ϕ. All vertices inside the gray bags are the same vertex of the graph. By case C.1.a, the blue subwalk $W^1[2, b-1]$ is palindrome, and therefore by Lemma 4.8, the vertices in it are unlabeled.

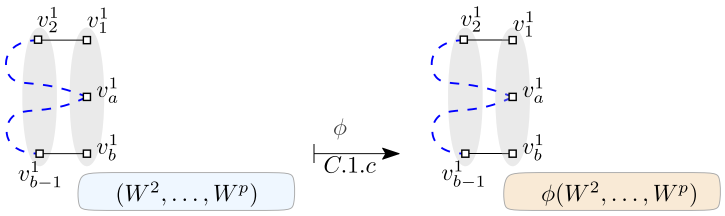


Fig. 12. Example of case C.1.c of the definition of ϕ. All vertices inside the gray bags correspond to the same vertex of the graph. By case C.1.a, the blue subwalk $W^1[2, b-1]$ is palindrome, and therefore by Lemma 4.8, the vertices in it are unlabeled.

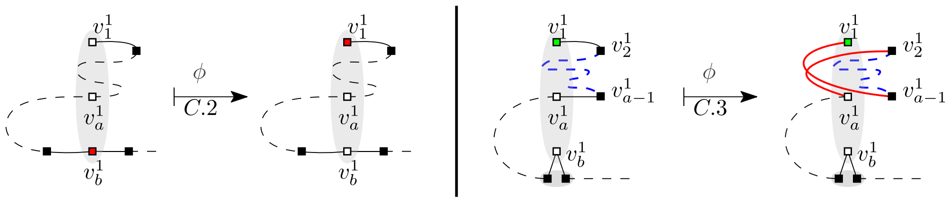


Fig. 13. Example of cases C.2 and C.3 of the definition of ϕ. Vertices inside the same gray bags correspond to the same vertex of the graph. On the left part of the figure (case C.2), in the initial configuration the vertex v_1^1 is unlabeled and the vertex v_b^1 is labeled (with color red) and the application of ϕ in this case exchanges this label from v_b^1 to v_1^1. On the right part of the figure, v_b^1 is a digon and therefore it is unlabeled and by cases C.1 and C.2, v_1^1 has to be labeled (depicted in green).

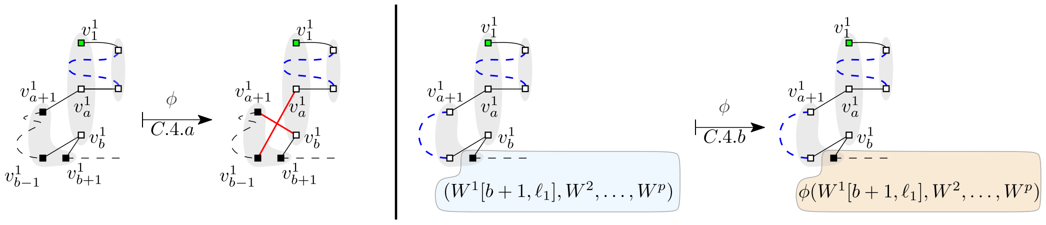

v_1^1
v_{a+1}^1
v_a^1
v_b^1
v_{b-1}^1
v_{b+1}^1
ϕ
$C.4.a$
$(W^1[b+1,\ell_1], W^2, \dots, W^p)$
$C.4.b$
$\phi(W^1[b+1,\ell_1], W^2, \dots, W^p)$

Fig. 14. Examples of cases C.4.a and C.4.b of the definition of ϕ. By case C.2, b is a digon on W^1 and by case C.3, $W^1[2, a-1]$ is a palindrome. For both case C.4.a and case C.4.b, we have that $v_{a+1}^1 = v_{b-1}^1$ (v_{a+1}^1, v_{b-1}^1, and v_{b+1}^1 are in the same grey bag). If $W^1[a+1, b-1]$ is not a palindrome, then we are in case C.4.a (on the left), while if $W^1[a+1, b-1]$ is a palindrome, we are in case C.4.b. (on the right).

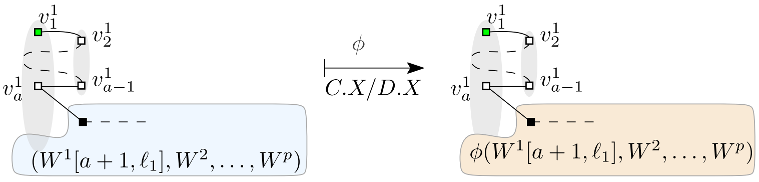

v_1^1
v_2^1
v_{a-1}^1
v_a^1
$(W^1[a+1,\ell_1], W^2, \ldots, W^p)$
ϕ
$C.X/D.X$
$\phi(W^1[a+1,\ell_1], W^2, \ldots, W^p)$

Fig. 15. Examples of cases C.X and D.X of the definition of ϕ.

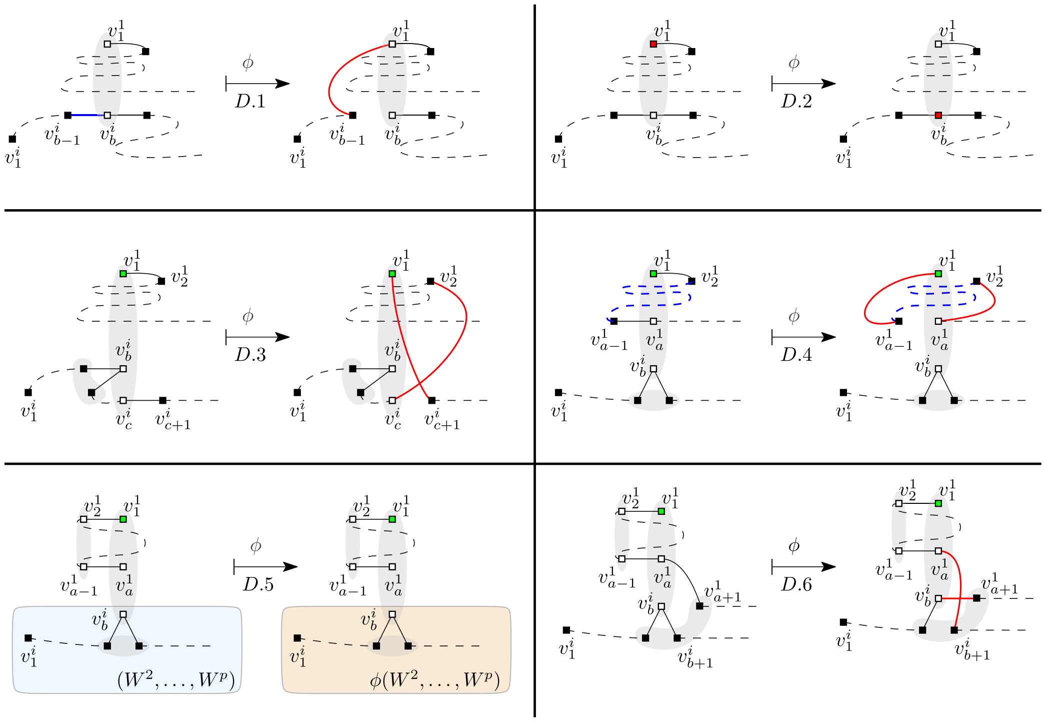


Fig. 16. Example of cases D.1-D.6 of the definition of ϕ. In each gray bag, all vertices inside the corresponding bag are the same vertex of the graph. Labeled vertices are depicted in red and green. White vertices correspond to unlabeled vertices and black vertices can be either labeled or unlabeled.

case, then the walkage $\phi(\mathcal{W})$ goes again to the same case, which then maps it back to $\mathcal{W}$. The only exception is that the cases C.X and D.X could map to each other.

Then, let us consider the cases relevant for a single walk, i.e., the case A.1 and the cases under C. Here, the intuition of case A.1 is to just move forward in the walk: we don't care much about what ϕ does to the rest of the walk because it must preserve the vertex right after v_1^1, and attaching v_1^1 to the front will not create a digon because v_1^1 occurs only at one index. Then, case C.1.a is the standard loop reversal case, which is safe because neither index 1 nor b is labeled. The case C.1.b (and C.1.c) corresponds to ignoring a palindromic subwalk, which can be safely done by Lemma 4.8. Then, case C.2 is the standard label swap case, which is safe because the index b is a not digon (note that the index 1 is never a digon). The cases C.1–C.2 are in some sense the "easy cases," while the cases C.3–C.X require more analysis of the remaining situation and quite unintuitive design. First, if neither C.1 nor C.2 applies, we know that the index 1 is labeled and the index b is digon. The purpose of case C.3 is to, in some sense reduce to a situation where we pretend that the vertex v_1^1 occurs only at indices 1, a, and b, as the walk between 1 and a is an irrelevant palindromic loop. Then, case C.4 handles a corner condition which would prevent case C.X from working, in particular, if $v_{a+1}^1 = v_{b-1}^1$ would hold in C.X, then it could potentially create a digon at the index $a + 1$. The case C.X ignores the palindromic loop between the indices 1 and a, leaving the only occurrence of the vertex v_1^1 in the rest of the walk to be at the digon b, which in some sense makes it "harmless" in that the recursive calls will never need to analyse the vertex v_b^1 again as the first vertex.

The intuition for the case of multiple walks is as follows. First, the case A.2 is just an analogue of A.1 when the first walk has length 1. Then, if the vertex v_1^1 occurs multiple times, we consider three different cases: v_1^1 occurs in at least three walks, v_1^1 occurs in one walk, and v_1^1 occurs in two walks. Here, the three walks case B is quite easy, as we can just consider two of the walks where v_1^1 is not labeled, circumventing all issues with labeled digons. When v_1^1 occurs in only one walk we go to the one walk case C. Then, when v_1^1 occurs in two walks W^1 and W^i, the intuition of cases under D is that we concatenate W^1 with reversed W^i, with some special marker in between, and then apply the single walk cases under C for this concatenation. Here, in the case D.X this can change whether v_1^1 occurs in two walks or a single walk, and therefore it is necessary to have the common case of C.X and D.X, moreover taking care in the proof that moving back from C.X to D.X will be handled correctly.

Correctness proof for ϕ. We will then proceed to first show that ϕ is well-defined, then that ϕ maps proper barren labeled walkages to proper barren labeled walkages, and then that ϕ satisfies all of the properties stated in Lemma 4.6, with $\phi(\phi(\mathcal{W})) = \mathcal{W}$ being the most complicated of them to prove. The proof is long because we have to analyze most of the 18 cases one by one. However, most of the arguments in these proofs are relatively easy once the definition of ϕ is set. The main challenge in the proof was to come up with the right definition of ϕ.

Well-definedness of ϕ. In Definition 4.9, in several cases, namely A.1, A.2, C.1.b, C.1.c, C.4.b, C.X, D.5, and D.X, the function ϕ is defined recursively. A priori it is not even clear why the syntactic value $\phi(\mathcal{W})$ is even well-defined in these cases. It requires proof that in these cases the recursive argument is in the domain of ϕ, in particular that it is also a proper barren labeled walkage.

Next we show that the syntactic value $\phi(\mathcal{W})$ for proper barren labeled walkages $\mathcal{W}$ is well-defined. We remark that Lemma 4.10 does not yet show that $\phi(\mathcal{W})$ is a proper barren labeled walkage; it will require more efforts to prove (see Lemmas 4.13, 4.12, and 4.11).

Lemma 4.10. *In case A.1 of Definition 4.9 it holds that $(W^1[2, \ell_1], W^2, \ldots, W^p)$ is a proper barren labeled walkage, in cases A.2, C.1.c, and D.5 it holds that $(W^2, \ldots, W^p)$ is a proper barren labeled walkage, in cases C.1.b, and C.4.b it holds that $(W^1[b + 1, \ell_1], W^2, \ldots, W^p)$ is a proper barren labeled walkage, and in cases C.X and D.X it holds that $(W^1[a + 1, \ell_1], W^2, \ldots, W^p)$ is a proper barren labeled walkage.*

Proof. In all cases, the labeled walkage used as the recursive argument is obtained from $\mathcal{W}$ by removing either the walk W^1 or a prefix of W^1. First we need to argue that the recursive argument is a labeled walkage. For this, the only thing to argue is that (1) the recursive argument contains at least one walk (i.e., $p \geq 2$ in cases A.2, C.1.c, and D.5) and that (2) all walks in the recursive argument are non-empty (i.e., $\ell_1 \geq 2$ in case A.1, $b < \ell_1$ in cases C.1.b and C.4.b, and $a < \ell_1$ in cases C.X and D.X). The other properties of labeled walkages are clearly satisfied when removing either W^1 or a prefix of W^1.

The above conditions are satisfied directly by definition in cases A.1 and C.1.b. In case C.4.b, $b < \ell_1$ holds by the fact that (due to case C.2) the index b is a digon in W^1. In case C.X, recall that a

is the index of the second last occurrence of v_1^1 in W^1, so $a < \ell_1$. In case D.X, we have that $a < \ell_1$ by case D.5. For the remaining cases A.2, C.1.c, and D.5, observe the following. If $p = 1$ would hold, then $\mathcal{W} = (W^1)$. Then, since in all these three cases $v_{\ell_1}^1 = v_1^1$ and $W^1[2, \ell_1]$ cannot contain labels (in A.2 trivially, in C.1.c by $r_1^1 = r_b^1 = 0$ and Lemma 4.8, and in D.5 by cases D.1, D.2, and D.4 combined with Lemma 4.8), it should hold that $(W^1[1, 1])$ is a labeled linkage consisting only of one walk with one vertex that would contradict the fact that $\mathcal{W}$ is barren by Definition 4.5.

It is clear by definition of a proper labeled walkage that removing a walk or a prefix of a walk maintains that the walkage is proper. To complete the proof, it remains to show case by case that the labeled walkages used as recursive arguments are barren.

In all of the cases the proofs will follow the same template: For the sake of contradiction we suppose that the labeled walkage $\mathcal{W}'$ used as a recursive argument is not barren, then consider the labeled linkage $\mathcal{W}''$ that witnesses that $\mathcal{W}'$ is not barren, and then use $\mathcal{W}''$ to construct a labeled linkage that shows that $\mathcal{W}$ is not barren, obtaining a contradiction. We spell out these steps in detail for the case A.1, and in less detail for subsequent cases. ◻

Case A.1. For the sake of contradiction, suppose that $\mathcal{W}' = (W^1[2, \ell_1], W^2, \ldots, W^p)$ is not barren. Then by the definition of barren, there exists a labeled linkage $\mathcal{W}''$ with $\texttt{start}(\mathcal{W}'') = \texttt{start}(\mathcal{W}')$, $\mathcal{T}(\mathcal{W}'') = \mathcal{T}(\mathcal{W}')$, $R(\mathcal{W}'') = R(\mathcal{W}')$, length $\leq \ell - 1$, and edges $E(\mathcal{W}'') \subseteq E(\mathcal{W}')$. By the assumptions of case A.1, the labeled walkage $\mathcal{W}'$ does not contain v_1^1, so the labeled linkage $\mathcal{W}''$ cannot contain v_1^1 because the edge property $E(\mathcal{W}'') \subseteq E(\mathcal{W}')$ ensures that v_1^1 cannot occur in a walk of length more than one, and the start vertex property $\texttt{start}(\mathcal{W}'') = \texttt{start}(\mathcal{W}')$ ensures that v_1^1 cannot occur in a walk of length one. By the start vertex property, it holds that the first vertex of the first walk in $\mathcal{W}''$ is v_2^1. Therefore, $W^1[1, 1] \diamond \mathcal{W}''$ is a labeled linkage. Because $v_1^1 v_2^1 \in E(\mathcal{W})$ and $E(\mathcal{W}'') \subseteq E(\mathcal{W}') \subseteq E(\mathcal{W})$, we have that $E(W^1[1, 1] \diamond \mathcal{W}'') \subseteq E(\mathcal{W})$. Also, observe that because of $R(\mathcal{W}'') = R(\mathcal{W}')$, it holds that $R(W^1[1, 1] \diamond \mathcal{W}'') = R(\mathcal{W})$. Similarly, we observe that $\texttt{start}(W^1[1, 1] \diamond \mathcal{W}'') = \texttt{start}(\mathcal{W})$, $\mathcal{T}(W^1[1, 1] \diamond \mathcal{W}'') = \mathcal{T}(\mathcal{W})$, and the length of $\mathcal{W}''$ is at most ℓ. Therefore, $W^1[1, 1] \diamond \mathcal{W}''$ is a labeled linkage that according to Definition 4.5 contradicts the fact that $\mathcal{W}$ is barren.

Case A.2. Again, suppose that $\mathcal{W}' = (W^2, \ldots, W^p)$ is not barren and consider the witness $\mathcal{W}''$. Because v_1^1 does not occur in $\mathcal{W}''$, it holds that $W^1 \sqcup \mathcal{W}''$ is a labeled linkage that contradicts that $\mathcal{W}$ is barren.

Case C.1.b. Suppose that $\mathcal{W}' = (W^1[b + 1, \ell_1], W^2, \ldots, W^p)$ is not barren and consider the witness $\mathcal{W}''$. As, by definition of b in case C, v_1^1 occurs in $\mathcal{W}$ only in the subwalk $W^1[1, b]$, it cannot occur in $\mathcal{W}''$. Therefore $W^1[1, 1] \diamond \mathcal{W}''$ is a labeled linkage. It is easy to observe that $\texttt{start}(W^1[1, 1] \diamond \mathcal{W}'') = \texttt{start}(\mathcal{W})$, $\mathcal{T}(W^1[1, 1] \diamond \mathcal{W}'') = \mathcal{T}(\mathcal{W})$, and $E(W^1[1, 1] \diamond \mathcal{W}'') \subseteq E(\mathcal{W})$. Also, Lemma 4.8 implies that $R(W^1[1, b]) = \emptyset$ and therefore $R(W^1[1, 1] \diamond \mathcal{W}'') = R(\mathcal{W})$, therefore contradicting that $\mathcal{W}$ is barren.

Case C.1.c. This case is similar as the previous, in particular, Lemma 4.8 implies that $R(W^1) = \emptyset$. Therefore, if we assume that $\mathcal{W}' = (W^2, \ldots, W^p)$ is not barren and we take the labeled linkage $\mathcal{W}''$ that witnesses that $\mathcal{W}'$ is not barren, we can construct a labeled linkage $W^1[1, 1] \sqcup \mathcal{W}''$ that contradicts the fact that $\mathcal{W}$ is barren.

Case C.4.b. Assume that $\mathcal{W}' = (W^1[b + 1, \ell_1], W^2, \ldots, W^p)$ is not barren and take the labeled linkage $\mathcal{W}''$ that witnesses that $\mathcal{W}'$ is not barren. As v_1^1 occurs in $\mathcal{W}$ only in the subwalk $W^1[1, b]$, it cannot occur in $\mathcal{W}''$. Therefore $W^1[1, 1] \diamond \mathcal{W}''$ is a labeled linkage. Note that in this case $W^1[2, a - 1]$ is a palindrome, the index a of the walk W^1 is not labeled because the index 1 is labeled and $\mathcal{W}$ is proper, and $W^1[a+1, b-1]$ is a palindrome, and the index b of W^1 is not labeled. Therefore, by Lemma 4.8, $R(W^1[1,b]) = \{v_1^1\}$, and therefore $R(W^1[1,1] \diamond \mathcal{W}'') = R(\mathcal{W})$, and therefore we contradict the fact that $\mathcal{W}$ is barren.

Case C.X. In this case, the argument is less apparent because v_1^1 indeed occurs in $W^1[a+1,\ell_1]$. Again, we start by assuming that $\mathcal{W}' = (W^1[a+1,\ell_1], W^2, \dots, W^p)$ is not barren, and take the labeled linkage $\mathcal{W}''$ that witnesses that $\mathcal{W}'$ is not barren. Now, note that v_1^1 occurs in $\mathcal{W}'$ only as a single digon in $W^1[a+1,\ell_1]$. Therefore, v_1^1 cannot occur as a starting or ending vertex in $\mathcal{W}''$. Also, there is only one edge in $E(\mathcal{W}')$ incident to v_1^1, which then prevents v_1^1 occuring at any position in $\mathcal{W}''$, because any position containing v_1^1 would have to a digon, but $\mathcal{W}''$ is labeled linkage and thus does not contain digons. Therefore, we construct a labeled linkage $W^1[1,1] \diamond \mathcal{W}''$ and use the fact that $W^1[2,a-1]$ is palindrome with Lemma 4.8 to conclude that $R(W^1[1,1] \diamond \mathcal{W}'') = R(\mathcal{W})$, and to finally observe that $W^1[1,1] \diamond \mathcal{W}''$ satisfies also all the other needed properties to contradict the fact that $\mathcal{W}$ is barren.

Case D.5. Note that here we have that $r_1^1 \neq 0$, $v_1^1 = v_{\ell_1}^1$, and $W^1[2,\ell_1 - 1]$ is a palindrome. The arguments are similar to case C.X: The vertex v_1^1 does occur in the walkage $(W^2, \dots, W^p)$, but it occurs in it only a single time, which is a digon in the walk W^i. So suppose that $\mathcal{W}' = (W^2, \dots, W^p)$ is not barren and consider the witness $\mathcal{W}''$. By similar arguments as in case C.X, v_1^1 cannot occur in $\mathcal{W}''$. Therefore, we construct a labeled linkage $W^1[1,1] \sqcup \mathcal{W}''$ and use the fact that $W^1[2,a-1]$ is palindrome with Lemma 4.8 to conclude that $R(W^1[1,1] \sqcup \mathcal{W}'') = R(\mathcal{W})$, and to finally observe that $W^1[1,1] \sqcup \mathcal{W}''$ satisfies also all the other needed properties to contradict the fact that $\mathcal{W}$ is barren.

Case D.X. Note that here again, we have that $r_1^1 \neq 0$ and $W^1[2,a-1]$ is a palindrome, where a is the last occurrence of v_1^1 in W^1. The arguments are similar to case C.X: The vertex v_1^1 does occur in the walkage $(W^1[a+1,\ell_1], W^2, \dots, W^p)$, but it occurs in it only a single time, which is a digon in the walk W^i. So again suppose that $\mathcal{W}' = (W^1[a+1,\ell_1], W^2, \dots, W^p)$ is not barren, and consider the witness $\mathcal{W}''$. Again by arguments of C.X we have that v_1^1 cannot occur in $\mathcal{W}''$. Therefore, we again construct a labeled linkage $W^1[1,1] \diamond \mathcal{W}''$ and use the fact that W^1 $[2,a-1]$ is palindrome with Lemma 4.8 to conclude that $R(W^1[1,1] \diamond \mathcal{W}'') = R(\mathcal{W})$, and to finally observe that $W^1[1,1] \diamond \mathcal{W}''$ satisfies also all the other needed properties to contradict that $\mathcal{W}$ is barren.

The next three lemmas establish that $\phi(\mathcal{W})$ is a proper barren labeled walkage. In addition, Lemma 4.11 shows that ϕ satisfies the properties $f(\phi(\mathcal{W})) = f(\mathcal{W})$, $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$, and $\texttt{start}(\phi(\mathcal{W})) = \texttt{start}(\mathcal{W})$.

Lemma 4.11. *Let $\mathcal{W}$ be a proper barren labeled walkage. It holds that $\phi(\mathcal{W})$ is a labeled walkage,* $\texttt{start}(\phi(\mathcal{W})) = \texttt{start}(\mathcal{W})$, *$\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$, and $f(\phi(\mathcal{W})) = f(\mathcal{W})$.*

Proof. We prove the lemma by induction on the length of the walkage $\mathcal{W}$. Here, all of the cases should be easy to verify, so the arguments we provide will be terse.

Cases A.1 works directly by induction, in particular, we can use the induction assumptions that

- $\phi(W^1[2,\ell_1], W^2, \dots, W^p)$ is a labeled walkage,
- $\texttt{start}(\phi(W^1[2,\ell_1], W^2, \dots, W^p)) = \texttt{start}((W^1[2,\ell], W^2, \dots, W^p))$,
- $\mathcal{T}(\phi(W^1[2,\ell_1], W^2, \dots, W^p)) = \mathcal{T}((W^1[2,\ell], W^2, \dots, W^p))$, and
- $f(\phi(W^1[2,\ell_1], W^2, \dots, W^p)) = f((W^1[2,\ell], W^2, \dots, W^p))$,

to prove the same properties for $\mathcal{W}$. In particular, we use the start vertex property to ensure that the first vertex of the first walk of $\phi(W^1[2,\ell_1], W^2, \dots, W^p)$ is v_2^1, and therefore the first edge of the first walk of $W^1[1,1] \diamond \phi(W^1[2,\ell_1], W^2, \dots, W^p)$ is $v_1^1 v_2^1$.

Case A.2 works similarly to A.1. Case B works by the property that $v_a^i = v_b^j$, in particular observing that if both of the suffixes are non-empty, the suffix swap operation preserves the set of ending vertices $\mathcal{T}(\mathcal{W})$. Case C.1.a works because $v_1^1 = v_b^1$, and cases C.1.b and C.1.c work by similar induction as A.1. Case C.2 works because $v_1^1 = v_b^1$, in particular, even though the index of the label in the walk changes, the vertex variable or the label-color pair variable do not change because the vertex does not change. Case C.3 works because $v_1^1 = v_a^1$. Case C.4.a works because $v_a^1 = v_b^1$ and case C.4.b works by similar induction as A.1. Case C.X works again by induction. Case D.1 works because $v_1^1 = v_b^i$. Case D.2 works because $v_1^1 = v_b^i$, by the same argument as case C.2.

In case D.3, all other conditions work directly by $v_1^1 = v_c^i$, but we should pay attention to the ending vertices condition $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$, because it can happen that one of the suffixes is empty. As observed already in the definition, observe that at most one of the suffixes can be empty because $v_1^1 = v_c^i$ and W^1 and W^i have different ending vertices because $\mathcal{W}$ is proper. First, if $\ell_1 = 1$, then the ending vertex of W^i becomes $v_c^i = v_1^1 = v_{\ell_1}^1$, and the ending vertex of W^1 becomes $v_{\ell_i}^i$, so the condition holds. Second, if $\ell_i = c$, then the ending vertex of W^1 becomes $v_1^1 = v_c^i = v_{\ell_i}^i$, and the ending vertex of W^i becomes $v_{\ell_1}^1$, so the condition holds.

Case D.4 works because $v_1^1 = v_a^1$. Case D.5 works by induction. Case D.6 works because $v_{a+1}^1 = v_{b+1}^i$ and the suffixes are guaranteed to be non-empty. Case D.X works by induction. □

Note that because $f(\mathcal{W})$ and $\texttt{start}(\mathcal{W})$ determine $R(\mathcal{W})$, $E(\mathcal{W})$, and the length of $\mathcal{W}$ uniquely, Lemma 4.11 implies that $\phi(\mathcal{W})$ is a barren walkage (because $\mathcal{W}$ is barren). It remains to prove that $\phi(\mathcal{W})$ is proper, and to prove that $\phi(\mathcal{W})$ is proper the only remaining thing to prove is that $\phi(\mathcal{W})$ does not contain labeled digons (Lemma 4.13). In particular, the property $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$ guarantees that the ending vertices of $\phi(\mathcal{W})$ are distinct, and $f(\phi(\mathcal{W})) = f(\mathcal{W})$, which implies $R(\phi(\mathcal{W})) = R(\mathcal{W})$ guarantees that the labeled vertices of $\phi(\mathcal{W})$ have different colors (because $\mathcal{W}$ is proper).

We will make use of the following lemma that follows directly from Lemma 4.11.

Lemma 4.12. *If a vertex occurs exactly once in $\mathcal{W}$ and this occurrence is a digon, then this vertex also occurs exactly once in $\phi(\mathcal{W})$ and this occurrence is also a digon with the same adjacent vertices.*

Proof. Suppose that a vertex v occurs exactly once in $\mathcal{W}$ and this occurrence is a digon. Therefore, it cannot be a starting or ending vertex in $\mathcal{W}$. By Lemma 4.11, it holds that $\texttt{start}(\phi(\mathcal{W})) = \texttt{start}(\mathcal{W})$ and $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$ and therefore v cannot be a starting or ending vertex neither in $\phi(\mathcal{W})$. Then, since by Lemma 4.11, we have that $f(\phi(\mathcal{W})) = f(\mathcal{W})$, implying $E(\phi(\mathcal{W})) = E(\mathcal{W})$, the vertex v must have the exactly same adjacent vertices in $\phi(\mathcal{W})$ as in $\mathcal{W}$. □

Then we prove that $\phi(\mathcal{W})$ has no labeled digons.

Lemma 4.13. *Let $\mathcal{W}$ be a proper barren labeled walkage. The labeled walkage $\phi(\mathcal{W})$ has no labeled digons.*

Proof. We prove the lemma by induction on the length of the walkage $\mathcal{W}$. □

Case A.1. In this case, as $\phi(W^1[2,\ell_1], W^2, \dots, W^p)$ has no labeled digons by induction, the only potential place for a labeled digon could be index 2 of W^1. However, because v_1^1 occurs only once in $\mathcal{W}$ and therefore does not occur in $\phi(W^1[2,\ell_1], W^2, \dots, W^p)$, we have that the index 2 of W^1 cannot become a digon.

Case A.2. Trivially by induction.

Case B. Here, by the definition of v^i_a and v^j_b in case B, both v^i_a and v^j_b are unlabeled and $v^i_a = v^j_b$ holds, so if $\mathcal{W} \leftrightarrow^{i,j}_{a,b}$ would contain a labeled digon then also $\mathcal{W}$ would.

Case C.1.a. Here, the indices 1 and b of W^1 are not labeled so they cannot become labeled digons. For indices in $[2, b-1]$, note that if $i \in [2, b-1]$ would be a labeled digon in $W^1\overleftarrow{[2, b-1]}$, then $b+1-i$ would have been a labeled digon in W^1.

Case C.1.b. Potential places for labeled digons here are incides b and $b+1$ at W^1. However, b is not labeled so no labeled digon can be at b, and because v^1_b does not occur in $(W^1[b+1, \ell_1], W^2, \dots, W^p)$, it cannot occur in $\phi(W^1[b+1, \ell_1], W^2, \dots, W^p)$ and therefore $b+1$ cannot become labeled digon.

Case C.1.c. Trivially by induction.

Case C.2. The index 1 of W^1 is not digon by definition of digon, and the index b is not digon by definition of this case, so no labeled digons are created.

Case C.3. Here, the index 1 cannot be a digon by definition, and the index a of W^1 has $r^1_a = 0$ by case C.2, so they cannot become labeled digons. For indices in $[2, a-1]$, the same argument as in case C.1.a applies.

Case C.4.a. Neither index a nor b is labeled so they cannot become labeled digons, and for indices in $[a+1, b-1]$ the same argument as in case C.1.a applies.

Case C.4.b. Same argument as in C.1.b, and using that by case C.2, it holds that $r^1_b = 0$.

Case C.X. Here, the index a of W^1 cannot become a labeled digon because it is not labeled. For the index $a+1$ the argument is more complicated: First note that the vertex $v^1_a = v^1_b (= v^1_1)$ occurs only once in $(W^1[a+1, \ell_1], W^2, \dots, W^p)$ (as v^1_b). Also, by case C.2, b is a digon in W^1. Therefore, by Lemma 4.12, the vertex $v^1_b = v^1_a$ occurs in $\phi(W^1[a+1, \ell_1], W^2, \dots, W^p)$ only once, and this occurrence is a digon adjacent with vertex $v^1_{b-1} = v^1_{b+1}$. Because by Lemma 4.11 ϕ preserves the starting vertices, and by case C.4 it holds that $v^1_{a+1} \neq v^1_{b-1}$, it holds that the occurrence of v^1_b in $\phi(W^1[a+1, \ell_1], W^2, \dots, W^p)$ cannot be as the second vertex of the first walk. Therefore, the index $a+1$ of the first walk cannot be a labeled digon in $W^1[1, a] \diamond \phi(W^1[a+1, \ell_1], W^2, \dots, W^p)$.

Case D.1. As $r^1_1 = r^i_b = 0$ and $v^1_1 = v^i_b$, it holds that if $\mathcal{W} \leftrightarrow^{1,i}_{1,b}$ would contain a labeled digon then also $\mathcal{W}$ would.

Case D.2. The index 1 of W^1 cannot become a digon by definition of digon. The index b of W^i cannot become a digon by definition of this case.

Case D.3. Because $v^1_1 = v^i_c$, in this case if $\mathcal{W} \leftrightarrow^{1,i}_{2,c+1}$ would have a labeled digon at index 2 of W^1 or at index $c+1$ of W^i, then this same labeled digon would have existed also in $\mathcal{W}$. Then, no labeled digon can be created to index 1 of W^1 by definition of digon or to index c of W^i because $r^1_1 \neq 0$ and therefore $r^i_c = 0$ by case D.2.

Case D.4. The index 1 of W^1 cannot become a digon by definition, and the index a cannot become a labeled digon because it is not labeled because the index 1 is labeled. For indices in $[2, a-1]$, the same argument as in case C.1.a applies.

Case D.5. Trivially by induction.

Case D.6. In this case, by cases D.1 and D.2 we have that $r_1^1 \neq 0$ and $r_b^i = 0$. The former implies that $r_a^1 = 0$. Therefore, the index a of W^1 nor the index b of W^i cannot become labeled digons because they cannot become labeled. Then, if the index $a+1$ of W^1 would be a labeled digon in $\mathcal{W} \leftrightarrow_{a+1,b+1}^{1,i}$, then the index $b+1$ of W^i would have been a labeled digon in $\mathcal{W}$ because $v_a^1 = v_b^i$ (and symmetrically for $b+1$ of W^i).

Case D.X. Here, a similar argument as in C.X works: By Lemma 4.12, we know that v_b^i occurs only as a digon surrounded by $v_{b-1}^i = v_{b+1}^i$ in $\phi(W^1[a+1,\ell_1], W^2, \dots, W^p)$. Therefore, because ϕ maintains the starting vertices and $v_{a+1}^1 \neq v_{b+1}^i$, it holds that the index $a+1$ of W^1 cannot be a digon in $W^1[1,a] \diamond \phi(W^1[a+1,\ell_1], W^2, \dots, W^p)$. The index a of W^1 cannot become a labeled digon because it is not labeled.

The function ϕ is an involution. Now we have shown that ϕ is a function $\phi : \mathcal{B} \rightarrow \mathcal{B}$, where $\mathcal{B}$ is the set of all barren proper labeled walkages, and that $f(\phi(\mathcal{W})) = f(\mathcal{W})$, $\mathcal{T}(\phi(\mathcal{W})) = \mathcal{T}(\mathcal{W})$, and $\texttt{start}(\phi(\mathcal{W})) = \texttt{start}(\mathcal{W})$ hold. Next we show that ϕ is an involution on $\mathcal{B}$, i.e., $\phi(\phi(\mathcal{W})) = \mathcal{W}$ holds.

Lemma 4.14. *For any proper barren labeled walkage $\mathcal{W}$ it holds that $\phi(\phi(\mathcal{W})) = \mathcal{W}$.*

Proof. We use induction on the length of walkage $\mathcal{W}$. The structure of the proof is to show that in all cases except C.X and D.X, the walk $\phi(\mathcal{W})$ goes to the same case of Definition 4.9 as $\mathcal{W}$. Then, the cases C.X and D.X are treated together. □

Case A.1. In both $\mathcal{W}$ and $\phi(\mathcal{W})$ it holds that the v_1^1 occurs only once in the walkage and ϕ does not change the first vertex, so $\phi(\phi(\mathcal{W})) = \mathcal{W}$ holds by induction.

Case A.2. In this case the first walk of $\mathcal{W}$ is the same as the first walk of $\phi(\mathcal{W})$, so $\phi(\phi(\mathcal{W})) = \mathcal{W}$ holds by induction.

Case B. In this case, the function ϕ preserves the set of walks in which v_1^1 occurs, and moreover preserves the walk in which v_1^1 occurs as labeled (if it occurs as labeled in any walk). Therefore, the walkage $\phi(\mathcal{W})$ also goes to case B, and in case B the indices i, j selected for $\phi(\mathcal{W})$ are the same as selected for $\mathcal{W}$. The suffix swap operation also does not change the indices of the first occurrences of v_1^1 in W^i and W^j, so the indices a and b selected are the same. Then, the lemma follows by observing that $\mathcal{W} \leftrightarrow_{a,b}^{i,j} \leftrightarrow_{a,b}^{i,j} = \mathcal{W}$.

At this point, let us observe that $\mathcal{W}$ goes to cases C.1-4 if and only if $\phi(\mathcal{W})$ goes to cases C.1-4. This is because all of these cases maintain that v_1^1 occurs only in the walk W^1, but multiple times in the walk W^1. In particular, in the recursive cases this is maintained by the fact that v_1^1 does not occur in the recursive argument.

We also observe that $\mathcal{W}$ goes to cases D.1-6 if and only if $\phi(\mathcal{W})$ goes to cases D.1-6. This is because all of these cases maintain that v_1^1 occurs in exactly two different walks. In the cases D.1-4 and D.6 this is easy to observe since these are not recursive, and in the case D.5 this follows from the fact that the walk W^1 is not changed and that in this case v_1^1 occurs exactly once in $(W^2, \dots, W^p)$.

Case C.1. Note that going to the cases C.1.a, C.1.b, and C.1.c depends only on the last occurrence b of v_1^1, and the labels r_1^1 and r_b^1. None of these cases change these, so $\phi(\phi(\mathcal{W})) = \mathcal{W}$ holds in case C.1.a because $W^1\overleftarrow{\overleftarrow{[2,b-1]}} = W^1$ and reversing a subwalk does not change whether it is a palindrome, and by induction in cases C.1.b and C.1.c.

Case C.2. The order of the vertices in the walks and the fact that exactly one of r^1_1 and r^1_b is labeled is maintained, so if $\mathcal{W}$ goes to case C.2 then also $\phi(\mathcal{W})$ goes to C.2. Then, $\phi(\phi(\mathcal{W})) = \mathcal{W}$ because $\mathcal{W} \frown^{1,1}_{1,b} \frown^{1,1}_{1,b} = \mathcal{W}$.

Case C.3. It holds that $a < b$, so therefore reversing $W^1[2, a-1]$ does not change the fact that b is a digon in W^1. It also does not change the index a of the second last occurrence of v^1_1, nor the index b of the last occurrence of v^1_1, nor the fact that $r^1_1 \neq 0$, nor the fact that $W^1[2, a-1]$ is not a palindrome.

Case C.4.a. Because $v^1_{a+1} = v^1_{b-1}$, reversing $W^1[a+1, b-1]$ does not change the fact that b is a digon in W^1. Reversing $W^1[a+1, b-1]$ also does not change the index a of the second last occurrence of v^1_1, nor the index b of the last occurrence of v^1_1, nor the fact that $r^1_1 \neq 0$, nor the fact that $W^1[2, a-1]$ is a palindrome.

Case C.4.b. Going to the case C.4.b depends only on the subwalk $W^1[1, b]$ and on the vertex with index $b+1$ in W^1 (whether the index b is a digon). Clearly, ϕ does not change the subwalk $W^1[1, b]$ in this case. The vertex with index $b+1$ is not changed because by Lemma 4.11 the starting vertices are preserved by $\phi(W^1[b+1, \ell_1], W^2, \ldots, W^p)$, so the lemma holds by induction.

Before moving to the cases C.X and D.X, we handle the cases D.1-D.5.

Case D.1. In this case, the operation $\mathcal{W} \leftrightarrow^{1,i}_{1,b}$ does not change the two walks in which v^1_1 occurs, nor it changes the fact that the first occurrence of v^1_1 in W^i is at index b, nor that $r^1_1 = r^i_b = 0$. The lemma follows from the fact that $\mathcal{W} \leftrightarrow^{1,i}_{1,b} \leftrightarrow^{1,i}_{1,b} = \mathcal{W}$.

Case D.2. This case does not change the vertices of the walks, so it is maintained that v^1_1 occurs only in W^1 and W^i. It also does not change the fact that exactly one of r^1_1 and r^i_b is labeled or the fact the index b is not a digon in W^i, so the lemma follows from the fact that $\mathcal{W} \frown^{1,i}_{1,b} \frown^{1,i}_{1,b} = \mathcal{W}$.

Case D.3. This case does not change the index b of the first occurrence of v^1_1 in W^i or the index c of the second occurrence of v^1_1 in W^i, and neither does it change the fact that $r^1_1 \neq 0$. Because $c > b+1$, it also does not change that the index b is a digon in W^i. The lemma follows from the fact that $\mathcal{W} \leftrightarrow^{1,i}_{2,c+1} \leftrightarrow^{1,i}_{2,c+1} = \mathcal{W}$.

Case D.4. This case does not change the index b of the first occurrence of v^1_1 in W^i, nor that $r^1_1 \neq 0$, nor that b is a digon in W^i, nor that v^1_1 occurs only once in W^i. It also does not change the index a of the last occurrence of v^1_1 in W^i, or the fact that $W^1[2, a-1]$ is a palindrome so the lemma holds.

Case D.5. The vertex v^1_1 occurs exactly once in $(W^2, \ldots, W^p)$, so it is also maintained that v^1_1 occurs in exactly two walks. The walk W^1 is not changed, so it is maintained that $r^1_1 \neq 0$ and therefore $\phi(\mathcal{W})$ does not go to case D.1. By Lemma 4.12, we have that $\phi(\mathcal{W})$ does not go to case D.2, and again because v^1_1 occurs only once in $(W^2, \ldots, W^p)$ we have that $\phi(\mathcal{W})$ does not go to case D.3. As W^1 is not changed we have that $\phi(\mathcal{W})$ does not go to case D.4. Then, as case D.5 does not change the walk W^1, it is maintained that $a = \ell_1$, so $\phi(\mathcal{W})$ goes to case D.5.

Case D.6. This case does not change that $r^1_1 \neq 0$, so $\phi(\mathcal{W})$ does not go to case D.1. It also does not change the index b of the first occurrence of v^1_1 in W^i, and because $v^1_{a+1} = v^i_{b+1}$, it does not change that b is a digon in W^i, and therefore $\phi(\mathcal{W})$ does not go to case D.2. Because a is the last occurrence of v^1_1 in W^1, it also does not change that v^1_1 occurs in W^i only once, so $\phi(\mathcal{W})$ does not go to case D.3. It also does not change the index a of the last occurrence of v^1_1 in W^1 or the subwalk

$W^1[2, a-1]$, so $\phi(\mathcal{W})$ does not go to cases D.4. or D.5. Therefore, $\phi(\mathcal{W})$ goes to case D.6 with the same values of a, b, and i, so $\phi(\phi(\mathcal{W})) = \mathcal{W}$ holds because $\mathcal{W} \leftrightarrow^{1,i}_{a+1,b+1} \leftrightarrow^{1,i}_{a+1,b+1} = \mathcal{W}$.

Case C.X. We aim to prove that if $\mathcal{W}$ goes to case C.X, then $\phi(\mathcal{W})$ also goes to case C.X or to case D.X, with the same value of the index a, and therefore $\phi(\phi(\mathcal{W})) = \mathcal{W}$ will hold by induction, as in both cases ϕ is defined as $\phi(\mathcal{W}) = W^1[1, a] \diamond \phi(W^1[a+1, \ell_1], W^2, \ldots, W^p)$. First, it is maintained that v_1^1 occurs more than once, but in at most two walks, because v_1^1 occurs only once in $(W^1[a+1, \ell_1], W^2, \ldots, W^p)$. Therefore, $\phi(\mathcal{W})$ does not go to case A or B.

Suppose that v_1^1 occurs in $\phi(\mathcal{W})$ only in the walk W^1, i.e., goes to case C. We will show that $\phi(\mathcal{W})$ goes to case C.X. It is maintained that $r_1^1 \neq 0$, so $\phi(\mathcal{W})$ does not go to case C.1. Then, by Lemma 4.12 it is maintained that the last occurrence of v_1^1 must be a digon so $\phi(\mathcal{W})$ does not go to case C.2. Also, this case does not change the index a of the second last occurrence of v_1^1 nor the walk $W^1[2, a-1]$, so it is maintained that $W^1[2, a-1]$ is a palindrome and therefore $\phi(\mathcal{W})$ does not go to case C.3. Then, to argue that $\phi(\mathcal{W})$ does not go to case C.4, observe that because ϕ maintains the starting vertex, the vertex at the position $a+1$ is maintained. The vertices around the digon at the last occurrence of v_1^1 are maintained by Lemma 4.12, so therefore if $\mathcal{W}$ does not go to case C.4 then also $\phi(\mathcal{W})$ does not go to case C.4. Therefore $\phi(\mathcal{W})$ goes to case C.X, and as the walk $W^1[1, a]$ is maintained, it goes to this case with the same value of a, so the lemma holds by induction.

Then, suppose that v_1^1 occurs in $\phi(\mathcal{W})$ in two walks W^1 and W^i, i.e., goes to case D. We will show that $\phi(\mathcal{W})$ goes to case D.X. It is maintained that $r_1^1 \neq 0$, so $\phi(\mathcal{W})$ does not go to case D.1. Then, by Lemma 4.12 it must be that v_1^1 occurs in W^i only once and as a digon, and therefore $\phi(\mathcal{W})$ does not go to case D.2 or D.3. Now, it will hold that the index a of the last occurrence of v_1^1 in the walk W^1 of $\phi(\mathcal{W})$ is the same as the index a of the last occurrence of v_1^1 in the walk W^1 of $\mathcal{W}$. Therefore, the subwalk $W^1[1, a]$ will be the same in $\mathcal{W}$ and $\phi(\mathcal{W})$, and therefore $\phi(\mathcal{W})$ will not go to case D.4 because $\mathcal{W}$ did not go to case C.3. Then, because ϕ cannot turn a non-empty walk into an empty walk, it is maintained that the length of W^1 is more than a, so $\phi(\mathcal{W})$ cannot go to case D.5. Then, for case D.6 we again note that ϕ maintains the vertex at position $a+1$, and that the digon around the occurrence of v_1^1 outside of $W^1[1, a]$ is maintained by Lemma 4.12. Therefore, $\phi(\mathcal{W})$ goes to case D.X with the same value of a, so the lemma holds by induction.

Case D.X. We will show that if $\mathcal{W}$ goes to case D.X, then $\phi(\mathcal{W})$ also goes to case D.X or to case C.X, with the same value of the index a, and therefore $\phi(\phi(\mathcal{W})) = \mathcal{W}$ will hold by induction, as in both cases ϕ is defined as $\phi(\mathcal{W}) = W^1[1, a] \diamond \phi(W^1[a+1, \ell_1], W^2, \ldots, W^p)$. First, it is maintained that v_1^1 occurs more than once, so therefore $\phi(\mathcal{W})$ does not go to case A. Then, as v_1^1 occurs only once in $(W^1[a+1, \ell_1], W^2, \ldots, W^p)$, it can occur in at most two walks in $\phi(\mathcal{W})$, so $\phi(\mathcal{W})$ cannot go to case B.

Suppose that v_1^1 occurs in $\phi(\mathcal{W})$ only in the walk W^1, i.e., goes to case C. We will show that $\phi(\mathcal{W})$ goes to case C.X. It is maintained that $r_1^1 \neq 0$, so $\phi(\mathcal{W})$ does not go to case C.1. Then, because $W^1[1, a]$ contains all other occurrences of v_1^1 in $\mathcal{W}$ except the occurrence in W^i, it must be that now the last occurrence of v_1^1 in W^1 of $\phi(\mathcal{W})$ corresponds to the occurrence of v_1^1 in W^i of $\mathcal{W}$, in particular, by Lemma 4.12 the last occurrence of v_1^1 in W^1 of $\phi(\mathcal{W})$ must be a digon, and therefore $\phi(\mathcal{W})$ does not go to case C.2. By the same reasoning, it also must be that the the index a of the last occurrence of v_1^1 in W^1 of $\mathcal{W}$ is the same as the index a of the second last occurrence of v_1^1 in W^1 of $\phi(\mathcal{W})$, and therefore the subwalk $W^1[1, a]$ of $\phi(\mathcal{W})$ in case C is the same as the subwalk $W^1[1, a]$ of $\mathcal{W}$ in case D. Then, it follows that because $\mathcal{W}$ did not go to case D.4, $\phi(\mathcal{W})$ does not go to case C.3. Then, as the case D.X does not change the vertex at the index $a+1$ of W^1, it holds that the vertex at the index $a+1$ of W^1 is the same in $\mathcal{W}$ and $\phi(\mathcal{W})$. Also, by Lemma 4.12

the vertices around the digon of the last occurrence of v_1^1 are the same in $\mathcal{W}$ and $\phi(\mathcal{W})$, so $\phi(\mathcal{W})$ does not go to case C.4 because $\mathcal{W}$ did not go to case D.5. Therefore, $\phi(\mathcal{W})$ must go to case C.X, and we already reasoned that the index a is the same as for $\mathcal{W}$ in case D.X, so the lemma holds by induction.

Suppose that v_1^1 occurs in $\phi(\mathcal{W})$ in two different walks, i.e., goes to case D. We will show that $\phi(\mathcal{W})$ goes to case D.X. First, it is maintained that $r_1^1 \neq 0$, so $\phi(\mathcal{W})$ does not go to case D.1. Then, we note that the walk W^i that contains the other occurrence of v_1^1 may be different for $\phi(\mathcal{W})$ than $\mathcal{W}$. However, it is maintained that v_1^1 occurs only once outside of W^1, and by Lemma 4.12 that the other occurrence is a digon and the vertices around this digon are maintained. Therefore, $\phi(\mathcal{W})$ does not go to case D.2, nor to case D.3. Now, the index a of the last occurrence of v_1^1 in W^1 will be the same for $\phi(\mathcal{W})$ and $\mathcal{W}$ because $\phi(\mathcal{W})$ does not change the subwalk $W^1[1,a]$. Therefore, it is maintained that $W^1[2,a-1]$ is a palindrome, and therefore $\phi(\mathcal{W})$ does not go to case D.4. Then, because ϕ cannot turn a non-empty walk into an empty walk, it is maintained that the length of W^1 is more than a, so $\phi(\mathcal{W})$ cannot go to case D.5. Then, by the start vertex property of ϕ, the vertex at index $a+1$ of W^1 is also maintained, and by Lemma 4.12 the vertices around the digon of the other occurrence of v_1^1 are maintained, so $\phi(\mathcal{W})$ does not go to case D.6. Therefore, $\phi(\mathcal{W})$ goes to case D.X, with the same value of a, and therefore the lemma holds by induction.

Finally, we show that ϕ is fixed-point-free.

Lemma 4.15. *For any proper barren labeled walkage $\mathcal{W}$ it holds that $\phi(\mathcal{W}) \neq \mathcal{W}$.*

Proof. We prove this by induction on the length ℓ of the walkage. In the recursive cases A.1, A.2, C.1.b, C.1.c, C.4.b, C.X, D.5, and D.X this holds directly by induction. In cases B, D.1, D.3, and D.6, ϕ changes the suffixes of two walks, and at least one of the suffixes is non-empty. Because $\mathcal{W}$ is proper, the ending vertices of all walks in $\mathcal{W}$ are different, so the ending vertex of at least one of the walks involved in the suffix swap is changed (in fact the ending vertices of both of the walks change, but it is not necessary for this proof). In cases C.1.a, C.3, C.4.a, and D.4, ϕ reverses a non-palindromic subwalk so $\phi(\mathcal{W}) \neq \mathcal{W}$. In cases C.2 and D.2 ϕ changes a label from one position to another, so $\phi(\mathcal{W}) \neq \mathcal{W}$. □

This completes the proof of Lemma 4.6.

5 From Colored Graphs to Frameworks

In this section, we extend our results from weighted colored graphs to weighted frameworks, in particular, we prove Theorem 1.5 (recall that Theorem 1.5 implies Theorem 1.3), and then discuss even further extensions to frameworks where the matroid is not necessarily represented over a finite field.

5.1 Frameworks

We recall definitions related to frameworks.

Matroids. We refer to the textbook of Oxley [41] for the introduction to Matroid Theory.

Definition 5.1. A pair $M = (V, \mathcal{I})$, where V is a *ground set* and $\mathcal{I}$ is a family of subsets of V, called *independent sets of* M, is a *matroid* if it satisfies the following conditions, called *independence axioms*:

(1) $\emptyset \in \mathcal{I}$,
(2) if $X \subseteq Y$ and $Y \in \mathcal{I}$ then $X \in \mathcal{I}$,
(3) if $X, Y \in \mathcal{I}$ and $|X| < |Y|$, then there is $v \in Y \setminus X$ such that $X \cup \{v\} \in \mathcal{I}$.

An inclusion maximal set of $\mathcal{I}$ is called a *base*. We use $V(M)$ and $\mathcal{I}(M)$ to denote the ground set and the family of independent sets of M, respectively.

The *direct sum* of matroids M and N is the matroid whose ground set is the disjoint union of $V(M)$ and $V(N)$, and whose independent sets are the disjoint unions of an independent set of M with an independent set of N.

Let $M = (V, \mathcal{I})$ be a matroid. We use 2^V to denote the set of all subsets of V. A function $r\colon 2^V \to \mathbb{Z}_{\geq 0}$ such that for every $X \subseteq V$,

$$r(X) = \max\{|Y|\colon Y \subseteq X \text{ and } Y \in \mathcal{I}\},$$

is called the *rank function* of M. The *rank of* M, denoted $r(M)$, is $r(V)$; equivalently, the rank of M is the size of any base of M. A matroid $M' = (V, \mathcal{I}')$ is a *k-truncation* of $M = (V, \mathcal{I})$ if for every $X \subseteq V$, $X \in \mathcal{I}'$ if and only if $X \in \mathcal{I}$ and $|X| \leq k$.

We work with several particular types of matroids. A *uniform* matroid is defined by the ground set V and its rank r; every subset S of V of size at most r is independent. *Partition* matroids are the matroids that can be written as disjoint sums of uniform matroids. *Transversal* matroids arise from graphs. For a bipartite graph $G = (V \cup B, E)$ with all edges between V and B, we can define a matroid $M = (V, \mathcal{I})$ such that a set $S \subseteq V$ is independent if there exists a matching in G such that every vertex in S is an endpoint of a matching edge.

Matroid representations. Let $M = (V, \mathcal{I})$ be a matroid and let $\mathbb{F}$ be a field. An $r \times n$-matrix A is a *representation of M over* $\mathbb{F}$ if there is a bijective correspondence f between V and the set of columns of A such that for every $X \subseteq V$, $X \in \mathcal{I}$ if and only if the set of columns $f(X)$ consists of linearly independent vectors of $\mathbb{F}^r$. Equivalently, A is a representation of M if M is isomorphic to the *column* matroid of A, that is, the matroid whose ground set is the set of columns of the matrix and the independence of a set of columns is defined as the linear independence. If M has a such a representation, then M is *representable* over $\mathbb{F}$ and it is also said M is a *linear* (or $\mathbb{F}$-*linear*) matroid. We can assume that the number of rows $r = r(M)$ for a matrix representing M [37].

Whenever we consider a linear matroid, it is assumed that its representation is given and the size of M is $\|M\| = \|A\|$, that is, the bit-length of the representation matrix. Notice that given a representation of a matroid, deciding whether a set is independent demands a polynomial number of field operations. In particular, if the considered field is a finite or is the field of rationals, we can verify independence in time that is a polynomial in $\|M\|$.

Frameworks. A framework is a pair (G, M), where $M = (V, \mathcal{I})$ is a matroid whose ground set is the set of vertices of G, i.e., $V(M) = V(G)$. A weighted framework is a triple $(G, M, \texttt{we})$, where (G, M) is a framework and $\texttt{we} : V(G) \to \mathbb{Z}_{\geq 1}$ is a weight function. An (S, T)-linkage $\mathcal{P}$ in a weighted framework $(G, M, \texttt{we})$ is (k, w)-ranked if $V(\mathcal{P})$ contains a set $X \subseteq V(\mathcal{P})$ with $X \in \mathcal{I}$, size $|X| = k$, and weight $\texttt{we}(X) = w$. When discussing algorithms for (weighted) frameworks, we explicitly specify how M is represented.

5.2 From Colored Graphs to Frameworks

In this section, we prove Theorem 1.5. We reduce the more general cases of matroids to the case of Theorem 1.4.

We start by giving our algorithm for the special case when the rank of M is bounded by k, in particular when M is represented as a $k \times n$ matrix.

Lemma 5.2. *There is a randomized algorithm, that given a weighted framework* $(G, M, \texttt{we})$, *where* G *is an* n*-vertex graph and* M *is represented as a* $k \times n$ *matrix over a finite field of order* q*, sets of vertices*

$S, T \subseteq V(G)$, and integers p, k, w, in time $2^{p+O(k^2 \log q)} n^{O(1)} w$ either finds a (k, w)-ranked (S, T)-linkage of order p and of minimum total length, or determines that $(G, M, \mathtt{we})$ has no (k, w)-ranked (S, T)-linkages of order p.

Proof. The matrix has at most q^k distinct column vectors so we can guess the k column vectors forming the independent set X of size k that we are looking for with at most q^{k^2} guesses. By inserting $|S|$ new vertices with neighborhoods equal to S, all-zero column vector, and weight 1, we can assume that the vectors of the starting vertices S will never correspond to the guessed basis. Then, we $k+1$-color the graph, assigning the color $k+1$ to the vertices of the set S and other vertices whose column vectors are not in the guessed basis, and the colors $[k]$ to the other vertices according to which of the k guessed column vectors they correspond to. We also assign the weight of all vertices whose column vector is not in the guessed basis to be 1.

Then, $(G, M, \mathtt{we})$ has a (k, w)-ranked (S, T)-linkage of order p if and only if it has a $(k+1, w+1)$-colored (S, T)-linkage of order p. In particular, the extra color $k+1$ contributes weight one and one color more, and the selected set $X \subseteq V(\mathcal{P})$ without the extra color must correspond to an independent set of M. Therefore, we get an algorithm with time complexity $q^{k^2} 2^{p+k} n^{O(1)} w = 2^{p+O(k^2 \log q)} n^{O(1)} w$. □

By extending Lemma 5.2 to matrices with a large number of rows by using randomized lossy truncation, we prove Theorem 1.5.

Proof of Theorem 1.5. Let A be a $r \times n$ matrix representing M. Our goal is to obtain a "lossy" representation of the k-truncation of M as a $k \times n$ matrix over a field of order $O(q+k^2)$. In particular, a representation so that any independent set of M of size k is independent in the representation with probability $\geq 1/2$, and any dependent set of M is dependent in the representation. Then, we obtain the algorithm by applying the algorithm of Lemma 5.2. Note that $2^{p+O(k^2 \log(q+k^2))} n^{O(1)} w = 2^{p+O(k^2 \log(q+k))} n^{O(1)} w$.

We use two techniques from [37], increasing the order of the field and truncation. First, we make sure that the order of the field is at least $2k$ by choosing the least integer i such that $q^i \geq 2k$, and going to the field of order q^i, as detailed in Proposition 3.2 of [37]. Now, we can assume that A is over a field of order at least $2k$ and at most $O(q+k^2)$.

Then, we truncate the matroid by multiplying the matrix A by a random $k \times r$ matrix R, in particular we claim that the $k \times n$ matrix $B = RA$ is now the desired representation of the k-truncation of M. The analysis here is the same as in Proposition 3.7 of [37], but with a smaller field. In particular, let us consider a subset U of the ground set of M, and let A_0 be the $r \times |U|$ submatrix of A corresponding to S. Now, in B, the $k \times |U|$ submatrix corresponding to S will be the matrix $B_0 = RA_0$. The rank of B_0 is at most the rank of A_0, so if U is dependent in M it will be dependent in the representation by B. Then, assume that U is an independent set of M and $|U| = k$. Now, $\det RA_0$ can be considered as a degree-k polynomial, that is not the zero polynomial, whose variables are the kr random entries of R. Therefore, by Lemma 3.1, the probability that $\det RA_0 = 0$ is at most $k/2k$. □

With minor adjustments, Theorem 1.5 can be adapted for frameworks with matroids that are in general not representable over a finite field of small order. For example, uniform matroids, and more generally transversal matroids, are representable over a finite field, but the field of representation must be large enough. We first show how Theorem 1.1 can be applied in the case of transversal matroids.

Theorem 5.3. *There is a randomized algorithm that given a weighted framework $(G, M, \mathtt{we})$, where G is an n-vertex graph and M is a transversal matroid represented by the corresponding bipartite* *graph, sets of vertices $S, T \subseteq V(G)$, and integers p, k, w, in time $2^{p+O(k^2 \log k)} n^{O(1)} w$ either finds a (k, w)-ranked (S, T)-linkage of order p and of minimum total length, or determines that $(G, M, \mathtt{we})$ has no (k, w)-ranked (S, T)-linkages of order p.*

Proof. We will construct a representation of the transversal matroid as a linear matroid over a finite field of order $O(k)$, so that any independent set of M of size k is independent in the representation with probability $\geq 1/2$, and any dependent set of M is dependent in the representation. This yields the algorithm by then using Theorem 1.5.

Our construction is the same as the construction of [37], except by using a smaller field. We choose the least prime p with $p \geq 2k$ and work in the field of order p. Let the bipartition of the vertices of the bipartite graph be (A, B). We construct an $|B| \times |A|$ matrix, so that an entry of the matrix is a random element of the field if it corresponds to an edge, and zero otherwise. Now, the determinant of a submatrix is guaranteed to be zero if there is no corresponding matching, so any dependent set of M is dependent in the representation. Otherwise, the determinant of a $k \times k$ submatrix can be seen as a degree-k polynomial (which is not the zero polynomial) that was evaluated for an uniformly random assignment of values to the variables. Therefore, as $p \geq 2k$, by Lemma 3.1, the probability that it is not zero is at least $1/2$. $\square$

It is also possible to apply Theorem 1.5 in the situation when M is represented by an integer matrix over rationals with entries bounded by $n^{O(k)}$.

Theorem 5.4. *There is a randomized algorithm that given a weighted framework $(G, M, \mathtt{we})$, where G is an n-vertex graph and M is represented as an integer matrix over rationals with entries bounded by $n^{O(k)}$, sets of vertices $S, T \subseteq V(G)$, and integers p, k, w, in time $2^{p+O(k^2 \log k)} n^{O(1)} w$ either finds a (k, w)-ranked (S, T)-linkage of order p and of minimum total length, or determines that $(G, M, \mathtt{we})$ has no (k, w)-ranked (S, T)-linkages of order p.*

Proof. Let c be a constant so that the entries of the matrix are bounded by n^{ck}. We pick a random prime p among the first $2 \log_2(k! n^{ck^2})$ primes, go to the finite field of order p by taking every entry modulo p, and then apply the algorithm of Theorem 1.1.

We first analyze the time complexity and then the correctness. By the prime number theorem, the prime p is bounded by

$$p = O(\log k! n^{ck^2} \cdot \log \log k! n^{ck^2}) = O(k^3 \log n \log \log n).$$

We can find such random prime in $n^{O(1)}$ time by elementary methods. Then, the time complexity by using Theorem 1.1 will be $2^{O(p+k^2 \log(k+k^3 \log n \log \log n))} n^{O(1)}$. Denote $t(n) = \log n \log \log n$ and consider two cases. First, if $t(n) \leq k^5$, then the time complexity is $2^{O(p+k^2 \log(k^8))} n^{O(1)} = 2^{O(p+k^2 \log k)} n^{O(1)}$. Second, if $t(n) > k^5$, then the time complexity is $2^{O(p+k^2 \log k^3 t(n))} n^{O(1)} = 2^{O(p+t(n)^{1/2} \log t(n))} n^{O(1)} = 2^{O(p)} \cdot 2^{O(\log^{1/2} n \log^{O(1)} \log n)} = 2^{O(p)} n^{O(1)}$.

Then, for the correctness we show that any dependent set of M is dependent in the representation and any independent set of M of size k is independent in the representation with probability $\geq 1/2$. Let A be a square submatrix of the original representation and A_p the corresponding submatrix in the presentation modulo p. Now, $\det A_p = \det A \mod p$. Therefore, all dependent sets stay dependent. Then, assume that A is a $k \times k$ submatrix corresponding to an independent set, i.e., $\det A \neq 0$. Now, the independent set can change into dependent only if $\det A$ is divisible by p. The value $\det A$ is bounded by $k! n^{ck^2}$, so there are at most $\log_2(k! n^{ck^2})$ primes dividing it. We chose p randomly among the first $2 \log_2(k! n^{ck^2})$ primes, so with probability $\geq 1/2$ the prime p does not divide $\det A$. $\square$

6 Deterministic Algorithm for Longest (S, T)-Linkage

This section is dedicated to the proof of Theorem 1.2.

We start with showing the main combinatorial lemma behind the theorem. The lemma is illustrated in Figure 3.

Lemma 6.1. *Let G be a digraph and let $C_1, \ldots, C_q$ be disjoint sets in $V(G)$. For $s, t \in V(G)$ let $P_1, \ldots, P_q$ be internally-disjoint (s,t)-paths. For each $i \in [q]$, let $v_i \in V(P_i)$ be such that the suffix of P_i starting from v_i lies inside C_i, except for t. For each $i \in [q]$, let Q_i be a path from v_i to t with all internal vertices in C_i. Then there exist internally-disjoint (s,t)-paths $P'_1, \ldots, P'_q$ such that P'_i is either (i) P_i or (ii) a composition of a prefix of P_i not containing any vertices of P_i beyond v_i, and a suffix of Q_j for some $j \in [q]$, and there is at least one path of type (ii) among $P'_1, \ldots, P'_q$.*

Proof. For each $i \in [q]$, denote the subpath of P_i from s to v_i by $P_i^{\rightarrow}$, and from v_i to t by $P_i^{\leftarrow}$. First, assume there exists $i \in [q]$ such that Q_i does not share a common internal vertex with any $P_j^{\rightarrow}$, $j \in [q]$. In this case, the solution is immediate: for each $j \neq i$, set $P'_j = P_j$, and set $P'_i = P_i^{\rightarrow} \circ Q_i$. The path $P_i^{\rightarrow}$ does not intersect any other P'_j internally since $P_1, \ldots, P_q$ are internally-disjoint, and by the assumption Q_i internally intersects neither $P_i^{\rightarrow}$ nor any other P'_j. So for the remaining part of the proof we assume that for each Q_i there exists $j \in [q]$ such that Q_i and $P_j^{\rightarrow}$ share a common internal vertex.

We now show the statement by analyzing a certain token sliding game. Intuitively, we put a token on each of the paths $P_1, \ldots, P_q$, originally on the place of the first intersection between P_i and some Q_j (see Figure 3(b)). Then we slide the tokens further along the paths according to certain rules, until the tokens reach a state where no rules can be applied (Figure 3(c) and (d)). Our goal is to show that in this case we obtain the desired paths $P'_1, \ldots, P'_q$.

More formally, we define a *stateS* as a tuple $(t_1, \ldots, t_q)$, where $t_i \in V(P_i)$ for each $i \in [q]$. The original state $S^1 = (t_1^1, \ldots, t_q^1)$ is defined as follows: for each $i \in [q]$, among all vertices of P_i that belong to Q_j for some $j \in [q]$, t_i^1 is the one closest to s. The game then proceeds iteratively, constructing the state S^{h+1} from S^h for each h starting from $h = 1$ by applying one of the following rules. For $j \in [q]$, we shall refer to a path Q_j as *active* if $t_j^h \neq t$.

Clear: Let $t_i^h = v_i$ for some $i \in [q]$. Then set $t_i^{h+1} = t$, and for each $j \in [q]$ such that $t_j^h \in V(Q_i) \setminus \{t\}$, set t_j^h to be the next vertex along the path P_j that belongs to an active $Q_{j'}$ for some $j' \in [q]$, here Q_i is not considered active. For all remaining $j \in [q]$, set $t_j^{h+1} = t_j^h$.

Push: Let i and i', $i \neq i' \in [q]$, be such that both t_i^h and $t_{i'}^h$ belong to $Q_j - \{t\}$ for some $j \in [q]$; additionally, let t_i^h be the farthest of two from t along Q_j. Set t_i^{h+1} to be the next vertex along the path P_i that belongs to an active $Q_{j'}$ for some $j' \in [q]$. For all $i'' \in [q]$, $i'' \neq i$, set $t_{i''}^{h+1} = t_{i''}^h$.

Intuitively, in the *Clear* rule, if at any step h the current token t_i^h of the path P_i reaches the vertex v_i, we forfeit this path: the token is moved to t, which corresponds to the ith path of the shorter solution being exactly P_i, and all other tokens on Q_i are moved next along their paths, until they hit another $Q_{j'}$. As for the *Push* rule, if two tokens end up on the same Q_j for some $j \in [q]$, we move the farthest of them from t further along its path P_i, similarly to the *Clear* rule.

As long as there is a possibility, a *Clear* rule is applied; *Push* is only applied if no *Clear* is available. If there are several options for applying the same rule, ties are breaking arbitrarily. We observe that every application of each rule moves at least one of the state vertices further along its respective path, and these vertices are never moved back. Thus, after a finite number of steps we reach a state where neither of the rules is applicable. Denote this state by S_i^T, our goal is to show the following.

Claim 6.2. *Let $S_i^T = (t_1^T, \ldots, t_q^T)$. There is $i \in [q]$ such that $t_i^T \neq t$.*

Before showing the proof of Claim 6.2 we make the following simple observation.

Claim 6.3. *For a state S^h, $h \in [T]$, for each $j \in [q]$, $t_j^h \in V(P_j)$, and t_j^h is either t or belongs to an active Q_i for some $i \in [q]$. Moreover, all of t_j^h that are not t, are distinct.*

Proof. By construction, the first part of the statement holds for the starting state S^1. Both rules either move vertices to an active Q_i further along its path, or directly to t. The second part follows immediately from the fact that the paths P_j are internally-disjoint. □

We first explain how Claim 6.2 implies the claim in the lemma. By Claim 6.2, there is at least one $i \in [q]$ such that $t_i^T \neq t$; let $I \subseteq [q]$ be the set of all indices with this property. By Claim 6.3, each vertex in $\{t_i^T\}_{i\in I}$ lies on an active Q_j for some $j \in [q]$, and these vertices are all distinct. Moreover, since the rule *Push* is not applicable, no two of these vertices share the same Q_j. Let $\pi : I \longrightarrow [q]$ be the injection that maps $i \in I$ to the index j such that $t_i^T \in V(Q_j)$. We construct the desired family of paths as follows: for $i \in [q] \setminus I$, let P'_i be P_i, and for $j \in I$, let P'_j be a concatenation of the subpath of P_i from s to t_j^T (denoted $\widehat{P}_i$), and the subpath of $Q_{\pi(i)}$ from t_j^T to t (denoted $\widehat{Q}_i$). Observe also that for each $i \in I$, t_i^T is not v_i since the rule *Clear* is not applicable. Moreover, since I is non-empty, there is at least one path of type (ii) in the constructed family. It only remains to show that the paths $\{P'_i\}_{i\in[q]}$ are internally-disjoint.

For $i \in [q] \setminus I$, denote $\widehat{P}_i = P_i^{\rightarrow}$ and $\widehat{Q}_i = P_i^{\leftarrow}$. For $i \neq i' \in [q]$, s is the only intersection between $\widehat{P}_i$ and $\widehat{P}_{i'}$ since $\widehat{P}_i$ and $\widehat{P}_{i'}$ are proper prefixes of P_i and $P_{i'}$ respectively, and these paths are internally disjoint by the assumption of the lemma. Observe that for each $i \in [q]$, the vertices of $\widehat{Q}_i$ except t lie in the set C_j, for some $j \in [q]$, and this correspondence between $\{\widehat{Q}_i\}_{i\in[q]}$ and $\{C_j\}_{j\in[q]}$ is a bijection defined by π on I and by the identity permutation on $[q] \setminus I$. Since the sets $\{C_j\}_{j\in[q]}$ are disjoint, for any $i \neq i' \in [q]$, we get that the paths $\widehat{Q}_i$ and $\widehat{Q}_{i'}$ share the only common vertex t.

It remains to verify that for each $i \neq i' \in [q]$, $\widehat{P}_i$ shares no common vertices with $\widehat{Q}_{i'}$. For $i' \in [q] \setminus I$, $\widehat{Q}_{i'} = P_{i'}^{\leftarrow}$, which is a suffix of $P_{i'}$, and this path cannot intersect $\widehat{P}_i$ which is a prefix of P_i; thus in the following we assume $i' \in I$. Assume the contrary, then there is a vertex u on $\widehat{P}_i$ that belongs to $Q_{\pi(i')}$, and is located on $Q_{\pi(i')}$ closer to t than $t_{i'}^T$, which is the starting vertex of $\widehat{Q}_{i'}$. Observe that the rule *Clear* has never been applied to $Q_{\pi(i')}$, otherwise $Q_{\pi(i')}$ would not be active in S^T. Thus, there is a state S^h where $t_i^h = u$, since any application of the rules to $t_i^{h'}$ for any $h' \in [T]$ either leaves this vertex in place or moves it to the next vertex of P_i belonging to an active Q_j for some $j \in [q]$. Now observe that no application of the rule *Push* makes the closest vertex to t on $Q_{\pi(i')} - \{t\}$ among the vertices of $\{t_j^h\}_{j\in[q]}$ farther, by definition of *Push*; or, in other words, the "leading" token on $Q_{\pi(i')} - \{t\}$ only gets closer to t with further steps. However, we get that t_i^h is closer to t on $Q_{\pi(i')}$ than $t_{i'}^T$, but $t_{i'}^T$ is the only vertex of $\{t_j^T\}_{j\in[q]}$ on $Q_{\pi(i')} - \{t\}$. This is a contradiction. Hence the assumption that $\widehat{P}_i$ and $\widehat{Q}_{i'}$ intersect cannot hold.

Proof of Claim 6.2. Assume the contrary, that $t_i^T = t$ for each $i \in [q]$. Since the paths $\{P_i^{\leftarrow}\}_{i\in[q]}$ are non-empty, $T > 1$. Thus, the state S^{T-1} is defined and S^T is obtained by applying a rule to S^{T-1}. First, observe that this rule could not have been *Push*, as it assumes there exist distinct t_i^{T-1} and $t_{i'}^{T-1}$ on $Q_j - \{t\}$ for some $i, i', j \in [q]$, and only moves t_i^{T-1} away while keeping $t_{i'}^T$ on $Q_j - \{t\}$. Therefore, *Clear* has been applied to S^{T-1}, replacing $t_i^{T-1} = v_j$ by $t_i^T = t$, for some $i, j \in [q]$. Now, we claim that there is another $i' \in [q]$, $i' \neq i$ such that $t_{i'}^{T-1} \in V(Q_j) \setminus \{t\}$. Indeed, by the starting assumption of the proof, there exists i' such that $P_{i'}^{\rightarrow}$ and Q_j share an internal vertex u. Since Q_j is active until the last step, and since the application of any rule moves $t_{i'}^h$ to the next vertex of $P_{i'}$ intersecting some active $Q_{j'}$, there exists a step $h \in [T-1]$ where $t_{i'}^h = u$. Before the step

$T-1$, only the rule *Push* could have been applied to Q_j, and an application of this rule never makes the intersection $\{t^h_{i''}\}_{i''\in[q]} \cap (Q_j \setminus \{v_j, t\})$ empty. Therefore there exists $i'' \in [q]$ such that $t^{T-1}_{i''} \in Q_j \setminus \{v_j, t\}$. This contradicts the assumption that $t^T_{i''} = t$ since the application of *Clear* to Q_j moves $t^{T-1}_{i''}$ to the next vertex on $P_{i''}$ on an active $Q_{j'}$; this will not take $t^T_{i''}$ farther than $v_{i''}$ along $P_{i''}$. □

Before we move to the proof of the main theorem, we note that the basic idea of random separation is to exploit random colorings of the vertex set. We, on the other hand, are first and foremost looking for a deterministic algorithm; the standard approach would be to enumerate a sufficiently "expressive" set of colorings, instead of trying a pre-set number of random colorings. Unfortunately, the existing results on derandomization of random separation algorithms cannot be applied directly, as normally random separation is considered for constant number of sets; most often two. Thus in the next lemma we directly construct a suitable family of functions by using the standard tool of perfect hash families, given by the classical result of Naor et al. [39] (we refer to [14, Chapter 5] for the detailed introduction to the concept). For integers n and k, an (n,k)*-perfect hash family*$\mathcal{F}$ is a family of functions from $[n]$ to $[k]$ such that for each set $S \subseteq [n]$ of size k there exists $f \in \mathcal{F}$ that acts on S injectively. We are now ready to state our derandomization lemma.

Lemma 6.4. *For an n-element set U and integers q, ℓ, there exists a family of functions $\mathcal{F}$ of size $q^{O(\ell)} \log n$ mapping U to $\{1,\dots,q\}$ with the following property. For any collection of disjoint sets $A_1,\dots, A_q \subseteq U$ with $\sum_{i=1}^{q} |A_i| \le \ell$, there exists a function $f \in \mathcal{F}$ such that $f(x) = i$ if $x \in A_i$. Moreover, $\mathcal{F}$ can be computed in time $q^{O(\ell)} n \log n$.*

Proof. First, for each integer $t \le \ell$ we construct an (n,t)-perfect hash family $\mathcal{H}_t$ of size $e^t t^{O(\log t)} \log n$ in time $e^t t^{O(\log t)} n \log n$ by the result of Naor, Schulman, and Srinivasan [39]. For each integer $t \le \ell$, every $h \in \mathcal{H}_t$ and a partition $[t] = I_1 \cup \ldots \cup I_q$, add a function $f^h_{I_1,\dots,I_q}$ to $\mathcal{F}$. The function acts as follows: for any $x \in U$, $f^h_{I_1,\dots,I_q}(x) = i$ if $h(x) \in I_i$.

We now show that $\mathcal{F}$ defined above satisfies the conditions of the lemma. Fix the subsets $A_1,\dots, A_q$ of U, denote $A = A_1 \cup \ldots \cup A_q$, and let $|A| = t \le \ell$. If $t = 0$, any function satisfies the requirement. Otherwise, by the definition of an (n,t)-perfect hash family, there exists $h \in \mathcal{H}_t$ such that the images $h(x)$ are distinct for all $x \in A$. For each $i \in [q]$, define I_i to be the set of indices that h assigns to A_i. By definition, $f^h_{I_1,\dots,I_q}(x) = i$ if $h(x) \in I_i$, and $h(x) \in I_i$ if and only if $x \in A_i$.

It remains to bound the size of $\mathcal{F}$. For each $t \le \ell$, the number of partitions $I_1,\dots, I_q$ of $[t]$ is at most q^t, since the total number of functions $[t] \to [q]$ is q^t. For each $t \le \ell$, $|\mathcal{H}_t| \le e^t t^{O(\log t)} \log n$, therefore $|\mathcal{F}| \le \sum_{t=1}^{\ell} |\mathcal{H}_t| \cdot q^t = q^{O(\ell)} \log n$. Clearly, $\mathcal{F}$ can be computed in time $q^{O(\ell)} n \log n$ as well. □

Finally, with Lemma 6.1 and Lemma 6.4 at hand, we move to the proof of Theorem 1.2 itself.

Proof of Theorem 1.2. First, we observe that finding an (S,T)-linkage of order p and total length at least k is equivalent to finding an (s,t)-linkage of order p and total length at least $(k+2)$, where moreover $s \ne t$ and s is not adjacent to t. Indeed, consider the digraph G' that is a copy of G with two new vertices s and t, where $N^+_{G'}(s) = S$ and $N^-_{G'}(t) = T$. Then, any directed (s,t)-linkage of order p and length $k+2$ in G' induces a directed (S,T)-linkage of order p and length k in G by removing s and t, and vice versa. Thus for the rest of the proof we assume that the task is to find an (s,t)-linkage of order p and total size at least k, $s \ne t$, and s is not adjacent to t. We now describe two separate subroutines of our algorithm, tailored for different cases of the maximum length of the path in the target (s,t)-linkage. The *short case* succeeds if there is an (s,t)-linkage where all paths have less than $2k$ internal vertices, and the *main case* succeeds otherwise (the proof of correctness follows after the description of the algorithm). □

Algorithm 1: Main Case of the Algorithm in Theorem 1.2

```
for $q' = 1, \ldots, p$ do
    Invoke Lemma 6.4 with $U = V(G) \setminus \{t\}$, $q = p + 1$, $\ell = 2kq$, to obtain the function family $\mathcal{F}$;
    foreach $f \in \mathcal{F}$ do
        Denote by $C_1, \ldots, C_{p+1}$ the vertices colored by the respective colors via $f$;
        for $i = 1, \ldots, q'$ do
            foreach $v_i \in C_i$ at distance $k$ from $t$ in $G_i = G[C_i \cup \{t\}]$ do
                Find a shortest $(v_i, t)$-path $Q_i$ in $G_i$;
                Find paths $P_1, P_2, ..., P_p$ in $G - (V(Q_i) \setminus \{v_i, t\})$, where $P_i$ is an $(s, v_i)$-path
                  and for each $j \neq i$, $P_j$ is an $(s, t)$-path, such that no two paths share a vertex
                  except for $s$ and $t$, and $P_i$ does not contain $t$;
                if such paths $P_1, P_2, ..., P_p$ exist then
                    Set $P_i = P_i \circ Q_i$;
                    return the paths $P_1, P_2, ..., P_p$;
                end
            end
        end
    end
end
```

Short case. For each $i \in [p]$, we branch over the number of internal vertices k_i of the ith path in the target linkage, $1 \leq k_i < 2k$. If $\sum_{i=1}^{p} k_i < k - 2$, we disregard the choice of $\{k_i\}_{i=1}^{p}$ and proceed to the next branch. Otherwise, consider a function family $\mathcal{F}$ given by an invocation of Lemma 6.4 with $U = V(G) \setminus \{s, t\}$, $q = p$ and $\ell = \sum_{i=1}^{p} k_i$. Branch over the choice of $f \in \mathcal{F}$ and denote by $C_1, C_2, \ldots, C_p$ the vertices colored by the respective colors via f. For each $i \in [p]$, we use a deterministic algorithm for finding a directed (s, t)-path with exactly k_i internal vertices in the graph $G[C_i \cup \{s, t\}]$ in time $2^{O(k_i)} \cdot n^{O(1)}$. The fastest-known such algorithm is the algorithm[3] of Zehavi [48] running in time $O(2.597^{k_i}) \cdot n^{O(1)}$. If for some choice of $\{k_i\}_{i=1}^{p}$ and f the desired collection of paths is found, the algorithm returns it. If no branch succeeds, the algorithm reports a no-instance.

We now argue for correctness of the algorithm above. Since the paths are internally disjoint by construction and $\sum_{i=1}^{p} k_i \geq k - 2$, if the algorithm returns a collection of paths, they clearly form a solution. In the other direction, fix a solution induced by directed (s, t)-paths $P_1^*, \ldots, P_p^*$, where for each $i \in [p]$ the ith path contains exactly $k_i < 2k$ internal vertices, and consider the respective branch of the algorithm above. If for each $i \in [p]$ the set C_i contains the internal vertices of the ith path, the algorithm succeeds, as $G[C_i \cup \{s, t\}]$ contains an (s, t)-path with exactly k_i internal vertices. Denote by A_i the set of internal vertices of P_i^*, for each $i \in [p]$, Lemma 6.4 guarantees that there exists $f \in \mathcal{F}$ that colors each A_i in color i, concluding the proof of correctness in this case.

Main case. The basic procedure is given in Algorithm 1. Observe that the task in Line 8 can be easily reduced to an instance of network flow, thus the whole procedure given in Lines 4 to 14 runs in polynomial time. If no iteration returns a collection of paths, the algorithm reports a no-instance.

[3]While the result in [48] is stated for finding an arbitrary path of certain length, it could be easily adjusted to finding an (s, t)-path.

It is easy to observe that if Algorithm 1 returns a collection of paths, then these paths constitute a solution to the given instance. Indeed, by construction, $P_1,\ldots,P_p$ are internally-disjoint (s,t)-paths. The length of Q_i is exactly k, since the path P_i contains Q_i as a subpath, the (s,t)-linkage given by the paths $P_1,\ldots,P_p$ is of order p and length at least k. It remains to verify that if there exists an (s,t)-linkage of order p where there is a path with at least $2k$ internal vertices, then our algorithm successfully returns a collection of paths for some choice of f. Denote the p-many (s,t)-paths that form a solution of minimum total length by $P^*_1,\ldots,P^*_p$, assuming that the paths are ordered from longest to shortest. In particular, $|V(P^*_1)\setminus\{s,t\}|\geq 2k$. Denote by q' the maximum index such that $|V(P^*_{q'})\setminus\{s,t\}|\geq 2k$, $1\leq q'\leq p$.

We say that the coloring $C_1,\ldots,C_p,C_{p+1}$ *agrees* with the paths $P^*_1,\ldots,P^*_p$ if the following holds:

(i) for each $i\in[q']$, the last k internal vertices of P^*_i belong to C_i,
(ii) the first k vertices of P^*_1 belong to C_{p+1},
(iii) for each $j\in[p]\setminus[q']$, all internal vertices of P^*_j belong to C_j.

For $i\in[q']$, denote by A_i the last k internal vertices of P^*_i; by A_{p+1} the first k vertices of P^*_1; for $j\in[p]\setminus[q']$, denote by A_j all internal vertices of P^*_j, we have that $|A_j|\leq 2k$. Since $\sum_{i=1}^{p+1}|A_i|\leq 2k(p+1)$, by Lemma 6.4, there exists $f\in\mathcal{F}$ that induces a coloring $C_1,\ldots,C_{p+1}$ with $A_i\subseteq C_i$ for each $i\in[p+1]$, meaning that this coloring agrees with the solution $P^*_1,\ldots,P^*_p$. In the remainder of the proof, we argue that for this choice of f Algorithm 1 outputs a solution.

For $i\in[q']$, let G_i denote the graph $G[C_i\cup\{t\}]$ and let v_i be the vertex of P^*_i at distance exactly k from t along the path. Since the coloring agrees with the solution, the last k internal vertices of P^*_i belong to C_i, including v_i. If v_i is at distance k from t in G_i, denote by Q_i the shortest (v_i,t)-path in G_i that the algorithm finds on Line 7; otherwise denote by Q_i an arbitrary shortest (v_i,t)-path in G_i. We now apply Lemma 6.1 to the paths $P^*_1,\ldots,P^*_{q'}$ in the graph G with selected disjoint vertex subsets $C_1,\ldots,C_{q'}$, with selected vertices $v_1,\ldots,v_{q'}$ and paths $Q_1,\ldots,Q_{q'}$. By the lemma, there exist internally disjoint (s,t)-paths $P'_1,\ldots,P'_{q'}$, such that for each $i\in[q']$, P'_i is either P^*_i or a concatenation of a prefix $\widehat{P}_i$ of P^*_i not extending beyond v_i, and a suffix $\widehat{Q}_i$ of Q_j for some $j\in[q']$; moreover, at least one of the paths is of the second type.

First, we claim that the paths $P'_1,\ldots,P'_{q'},P^*_{q'+1},\ldots,P^*_p$ together form an $(\{s\},\{t\})$-linkage of length at least k. Indeed, the paths $P^*_1,\ldots,P^*_p$ are internally-disjoint from the beginning; thus for each $q'<i<j\leq p$, P^*_i and P^*_j do not share common internal vertices. Moreover, for each $i\in[q']$, $j\in[p]\setminus[q']$, P'_i and P^*_j are immediately internally disjoint in case $P'_i=P^*_i$. In case P'_i is a concatenation of $\widehat{P}_i$ and $\widehat{Q}_i$, the path $\widehat{P}_i$ again does not share a vertex with P^*_j as a prefix of P^*_i, except for s. The suffix $\widehat{Q}_i-\{t\}$ on the other hand is fully contained in some $C_{i'}$, $i'\in[q']$, while $P^*_j-\{s,t\}$ is contained in C_j by the property (iii) of the coloring. Finally, for the length observe that the path P'_1 contains the first k vertices of P^*_1. This holds since by Lemma 6.1, P'_1 contains a prefix of P^*_1 that ends either in t, or in a vertex of C_j, for $j\in[q]$; on the other hand, C_{p+1} contains the first k vertices of P^*_1 and is disjoint from t and any C_j, $j\in[q]$, therefore these k vertices are also contained in P'_1.

Consider now a path P'_i that is not P^*_i, but a concatenation $\widehat{P}_i\circ\widehat{Q}_i$. The length of $\widehat{Q}_i$ is at most k, which is the length of the suffix of P^*_i from v_i to t, and $\widehat{P}_i$ is at most as long as the prefix of P^*_i from s to v_i. Therefore, the length of the path P'_i is at most the length of P^*_i. Since this holds for every $i\in[q']$, also when $P'_i=P^*_i$, the total length of $P'_1,\ldots,P'_{q'},P^*_{q'+1},\ldots,P^*_p$ is at most the total length of $P^*_1,\ldots,P^*_p$. By the previous argument, the former is an $(\{s\},\{t\})$-linkage of length at least k, while the latter by definition is such an $(\{s\},\{t\})$-linkage of smallest total length. Therefore, it must be the case that for each $i\in[q']$, the length of P'_i is equal to the length of P^*_i. In particular, if

P'_i is not P^*_1, but a concatenation $\widehat{P}_i \circ \widehat{Q}_i$, then $\widehat{P}_i$ is exactly the prefix of P^*_i from s to v_i, and $\widehat{Q}_i$ is Q_i; additionally, Q_i is then of length exactly k, thus v_i is at distance k from t in G_i, and Q_i is the shortest path chosen by the algorithm in Line 7.

We now claim that the existence of (s,t)-linkage $P'_1,\ldots, P'_{q'}, P^*_{q'+1},\ldots, P^*_p$ implies that Line 8 is executed successfully for the respective choice of i and v_i. Specifically, let $i \in [q']$ be such that the path P'_i is not P^*_i, but a concatenation $\widehat{P}_i \circ \widehat{Q}_i$, and let v_i be the common endpoint of $\widehat{P}_i$ and $\widehat{Q}_i$. Indeed, denote $P_i = \widehat{P}_i$, for each $i' \in [q']$, $i' \neq i$, denote $P_{i'} = P'_i$, and for each $j \in [p] \setminus [q']$, denote $P_j = P^*_j$. This collection of paths is of form required by the conditions in Line 8, and thus the algorithm of Line 8 returns a suitable collection of paths as well (not necessarily the same). This concludes the proof of correctness for *Main case*.

Finally, observe that the running time of both cases is dominated by the invocation of Lemma 6.4 with $q = p + 1$ and $\ell \leq 2k(p + 1)$, resulting in the total running time of $p^{O(kp)} \cdot n^{O(1)}$. □

7 Conclusion

We conclude with several concrete open questions. The first question is about derandomizing Theorem 1.1, even for the case when $p = 1$. The algorithm in Theorem 1.1 is based on DeMillo–Lipton–Schwartz–Zippel lemma for polynomial identity testing, and therefore we do not expect to derandomize it using similar techniques [27, 38]. However, similarly to Theorem 1.2, we do not exclude that other methods could result in (maybe slower) deterministic algorithms. We are not aware of any *deterministic* and FPT in k algorithm for Maximum Colored Path.

The second question is about the Disjoint Paths problem. Here for a given set of pairs of terminal vertices $(s_1, t_1), \ldots, (s_r, t_r)$, the problem is to decide whether there are vertex-disjoint (s_i, t_i)-paths, $i \in [r]$. The problem is FPT parameterized by r by the seminal algorithm of Robertson and Seymour [43]. A natural extension of this problem would be on colorful graphs, where we want the disjoint paths to collect at least k colors. We do not know whether the colored variant of the problem is FPT parameterized by k even for 2-Disjoint Paths, that is for $r = 2$.

The third question concerns extending Theorem 1.3, where we demand matroid M to be represented as a matrix over a finite field of order q. The natural question here is whether there is an FPT algorithm for k-ranked (s,t)-path (and more generally, for k-ranked (S,T)-linkage of order p) in frameworks (G, M), where M is a linear matroid represented as a matrix over rationals. We also ask what is the complexity of this problem when M is given by an independence oracle. As was shown by Jensen and Korte [26], various matroid problems have unconditional complexity lower bounds asserting that they admit no algorithms where the number of oracle calls is bounded by a polynomial on the size of the matroid ground set. For example, this concerns the classical Matroid Parity problem that can be solved in polynomial time on linear matroids as it was shown by Lovász (see, e.g., [36]). It is natural to ask whether such a lower bound can be shown for k-ranked (s,t)-path.

The last concrete question is about Longest (s,t)-Path and Longest Cycle. Our algorithm implies the first $2^k n^{O(1)}$ time algorithms for these problems, and the dependency on k in the time complexity of our algorithm is unlikely to be improved in the general colored case. However, it remains an interesting open problem whether Longest (s,t)-Path or Longest Cycle could be solved in time $(2 - \varepsilon)^k n^{O(1)}$ for some $\varepsilon > 0$, especially keeping in mind that k-Path admits an $1.66^k n^{O(1)}$ time algorithm [3].

Acknowledgments

We thank Shyan Akmal for remarking that combining [22] with [46] implies a $4^k n^{O(1)}$ time algorithm for Longest (s,t)-Path.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *J. ACM* 42, 4 (1995), 844–856. DOI: https://doi.org/10.1145/210332.210337

[2] Andreas Björklund. 2014. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.* 43, 1 (2014), 280–299. DOI: https://doi.org/10.1137/110839229

[3] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. 2017. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.* 87 (2017), 119–139. DOI: https://doi.org/10.1016/j.jcss.2017.03.003

[4] Andreas Björklund, Thore Husfeldt, and Nina Taslaman. 2012. Shortest cycle through specified elements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*. Yuval Rabani (Ed.), SIAM, 1747–1753. DOI: https://doi.org/10.1137/1.9781611973099.139

[5] Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. 2016. Constrained multilinear detection and generalized graph motifs. *Algorithmica* 74, 2 (2016), 947–967. DOI: https://doi.org/10.1007/s00453-015-9981-1

[6] Hajo Broersma, Xueliang Li, Gerhard J. Woeginger, and Shenggui Zhang. 2005. Paths and cycles in colored graphs. *Australas. J Comb.* 31 (2005), 299–312. http://ajc.maths.uq.edu.au/pdf/31/ajc_v31_p299.pdf

[7] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766. DOI: https://doi.org/10.1137/080733991

[8] Raffaele Cerulli, Paolo Dell'Olmo, Monica Gentili, and Andrea Raiconi. 2006. Heuristic approaches for the minimum labelling hamiltonian cycle problem. *Electron. Notes Discret. Math.* 25 (2006), 131–138. DOI: https://doi.org/10.1016/j.endm.2006.06.080

[9] Chandra Chekuri and Martin Pál. 2005. A recursive greedy algorithm for walks in directed graphs. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 245–253.

[10] N. Christofides. 1985. Vehicle routing. In *The Traveling Salesman Problem*. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (Eds.), Wiley, 431–448.

[11] Johanne Cohen, Giuseppe F. Italiano, Yannis Manoussakis, Nguyen Kim Thang, and Hong Phong Pham. 2021. Tropical paths in vertex-colored graphs. *J. Comb. Optim.* 42, 3 (2021), 476–498. DOI: https://doi.org/10.1007/s10878-019-00416-y

[12] Basile Couëtoux, Elie Nakache, and Yann Vaxès. 2017. The maximum labeled path problem. *Algorithmica* 78, 1 (2017), 298–318. DOI: https://doi.org/10.1007/s00453-016-0155-6

[13] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. 2016. On problems as hard as CNF-SAT. *ACM Trans. Algorithms* 12, 3 (2016), 41 1–41:24. DOI: https://doi.org/10.1145/2925416

[14] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, MichaÅ, Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer.

[15] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. 2013. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 301–310.

[16] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał, Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. 2011. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 150–159.

[17] Reinhard Diestel. 2012. *Graph theory* (4th ed.). Graduate texts in mathematics, Vol. 173. Springer.

[18] Eduard Eiben and Iyad Kanj. 2020. A colored path problem and its applications. *ACM Trans. Algorithms* 16, 4 (2020), 47 1–47:48. DOI: https://doi.org/10.1145/3396573

[19] Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. 2020. Algorithmic extensions of dirac's theorem. arXiv:2011.03619. Retrieved from https://arxiv.org/abs/2011.03619

[20] Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. 2022. Algorithmic extensions of dirac's theorem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA '22)*. SIAM, 406–416.

[21] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. 2016. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM* 63, 4 (2016), 29 1–29:60. DOI: https://doi.org/10.1145/2886094

[22] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. 2018. Long directed (s, t)-path: FPT algorithm. *Inf. Process. Lett.* 140 (2018), 8–12. DOI: https://doi.org/10.1016/j.ipl.2018.04.018

[23] Steven Fortune, John E. Hopcroft, and James Wyllie. 1980. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.* 10 (1980), 111–121. DOI: https://doi.org/10.1016/0304-3975(80)90009-2

[24] Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. 2011. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 479–488.

[25] Refael Hassin, Jérôme Monnot, and Danny Segev. 2007. Approximation algorithms and hardness results for labeled connectivity problems. *J. Comb. Optim.* 14, 4 (2007), 437–453. DOI: https://doi.org/10.1007/s10878-007-9044-x

[26] Per M. Jensen and Bernhard Korte. 1982. Complexity of matroid property algorithms. *SIAM J. Comput.* 11, 1 (1982), 184–190. DOI: https://doi.org/10.1137/0211014

[27] Valentine Kabanets and Russell Impagliazzo. 2004. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity* 13, 1 (2004), 1–46.

[28] Ken-ichi Kawarabayashi. 2008. An improved algorithm for finding cycles through elements. In *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*(Lecture Notes in Computer Science, Vol. 5035):cli. Springer, 374–384. DOI: https://doi.org/10.1007/978-3-540-68891-4_26

[29] Ioannis Koutis. 2008. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*. Lecture Notes in Computer Science, Vol. 5125, Springer, 575–586.

[30] Ioannis Koutis and Ryan Williams. 2016. Algebraic fingerprints for faster algorithms. *Commun. ACM* 59, 1 (2016), 98–105. DOI: https://doi.org/10.1145/2742544

[31] Lukasz Kowalik and Juho Lauri. 2016. On finding rainbow and colorful paths. *Theor. Comput. Sci.* 628 (2016), 110–114. DOI: https://doi.org/10.1016/j.tcs.2016.03.017

[32] Neeraj Kumar, Daniel Lokshtanov, Saket Saurabh, and Subhash Suri. 2021. A constant factor approximation for navigating through connected obstacles in the plane. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 822–839.

[33] Andrzej Lingas and Mia Persson. 2015. A fast parallel algorithm for minimum-cost small integral flows. *Algorithmica* 72, 2 (2015), 607–619. DOI: https://doi.org/10.1007/s00453-013-9865-1

[34] L. Lovász. 1977. Flats in matroids and geometric graphs. In *Combinatorial Surveys: Proceedings of the 6th British Combinatorial Conference*. 45–86.

[35] László Lovász. 2019. *Graphs and geometry*. American mathematical society colloquium publications, Vol. 65. American Mathematical Society, Providence, RI. x+444 pages. DOI: https://doi.org/10.1090/coll/065

[36] L. Lovász and M. D. Plummer. 1986. *Matching theory*. North-Holland mathematics studies, Vol. 121: Annals of discrete mathematics, Vol. 29. North-Holland Publishing Co., Amsterdam; North-Holland Publishing Co., Amsterdam. xxvii+544 pages.

[37] Dániel Marx. 2009. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.* 410, 44 (2009), 4471–4479. DOI: https://doi.org/10.1016/j.tcs.2009.07.027

[38] Juan Andrés Montoya and Moritz Müller. 2013. Parameterized random complexity. *Theory Comput. Syst.* 52, 2 (2013), 221–270. DOI: https://doi.org/10.1007/s00224-011-9381-0

[39] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. 1995. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*. IEEE, 182–191.

[40] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.* 14, 1 (1978), 265–294. DOI: https://doi.org/10.1007/BF01588971

[41] James Oxley. 2011. *Matroid theory* (2nd ed.). Oxford graduate texts in mathematics, Vol. 21. Oxford University Press, Oxford. xiv+684 pages. DOI: https://doi.org/10.1093/acprof:oso/9780198566946.001.0001

[42] Fahad Panolan, Saket Saurabh, and Meirav Zehavi. 2019. Parameterized algorithms for list k-cycle. *Algorithmica* 81, 3 (2019), 1267–1287. DOI: https://doi.org/10.1007/s00453-018-0469-7

[43] Neil Robertson and Paul D. Seymour. 1995. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B* 63, 1 (1995), 65–110. DOI: https://doi.org/10.1006/jctb.1995.1006

[44] Jacob T. Schwartz. 1980. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* 27, 4 (1980), 701–717. DOI: https://doi.org/10.1145/322217.322225

[45] Magnus Wahlström. 2013. Abusing the tutte matrix: An algebraic instance compression for the K-set-cycle problem. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS '13)*. Natacha Portier and Thomas Wilke (Eds.), Leibniz International Proceedings in Informatics, Vol. 2, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 341–352. DOI: https://doi.org/10.4230/LIPIcs.STACS.2013.341

[46] Ryan Williams. 2009. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.* 109, 6 (2009), 315–318.

[47] Hans-Christoph Wirth. 2001. *Multicriteria Approximation of Network Design and Network Upgrade Problems*. Ph.D. Dissertation. Universität Würzburg.

[48] Meirav Zehavi. 2015. Mixing color coding-related techniques. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA '15)*. Nikhil Bansal and Irene Finocchi (Eds.), Springer, Berlin, 1037–1049.

[49] Meirav Zehavi. 2016. A randomized algorithm for long directed cycle. *Inf. Process. Lett.* 116, 6 (2016), 419–422. DOI: https://doi.org/10.1016/j.ipl.2016.02.005

[50] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM '79)*. Edward W. Ng (Ed.), Lecture Notes in Computer Science, Vol. 72, Springer, 216–226. DOI: https://doi.org/10.1007/3-540-09519-5_73

Received 23 August 2023; revised 18 June 2024; accepted 18 June 2024