Theoretical Computer Science 795 (2019) 275–284

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Two edge-disjoint paths with length constraints [☆]

Leizhen Cai [1], Junjie Ye [2]

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong SAR, China

ARTICLE INFO

Article history:
Received 9 December 2017
Received in revised form 20 May 2019
Accepted 3 July 2019
Available online 8 July 2019
Communicated by G.F. Italiano

Keywords:
Edge-disjoint paths
Random partition
FPT algorithm
Kernelization

ABSTRACT

We consider the problem of finding, for two pairs (s_1, t_1) and (s_2, t_2) of vertices in an undirected graph, an (s_1, t_1)-path P_1 and an (s_2, t_2)-path P_2 such that P_1 and P_2 share no edges and the length of each P_i satisfies constraint L_i, where $L_i \in \{\leq k_i, = k_i, \geq k_i, *\}$ with $L_i =$ "$*$" indicating no length constraint on P_i.

We regard k_1 and k_2 as parameters and investigate the parameterized complexity of the above problem when at least one of P_1 and P_2 has a length constraint. For the 9 different cases of (L_1, L_2), we obtain FPT algorithms for 7 of them by using random partition backed by some structural results. On the other hand, we prove that the problem admits no polynomial kernel for all 9 cases unless $NP \subseteq coNP/poly$.

1. Introduction

Disjoint paths in graphs are fundamental and have been studied extensively in the literature. Given p pairs of *terminal vertices* (s_i, t_i) for $1 \leq i \leq p$ in an undirected graph G, the classical Edge-Disjoint Paths problem asks whether G contains p pairwise edge-disjoint paths P_i between s_i and t_i for all $1 \leq i \leq p$. The problem is NP-complete as shown by Even et al. [13], but is solvable in time $O(mn)$ by network flow [22] if all vertices s_i (resp., t_i) are the same vertex s (resp., t). When we regard p as a parameter, a celebrated result of Robertson and Seymour [23] on vertex-disjoint paths can be used to obtain an FPT algorithm for Edge-Disjoint Paths. On the other hand, Bodlaender et al. [6] have shown that the vertex-disjoint variation of Edge-Disjoint Paths admits no polynomial kernel unless $NP \subseteq coNP/poly$.

In this paper, we study Edge-Disjoint Paths with length constraint L_i on each (s_i, t_i)-path P_i and focus on the problem for two pairs of terminal vertices. The length constraint $L_i \in \{\leq k_i, = k_i, \geq k_i, *\}$ indicates that the length of P_i needs to satisfy L_i. In particular, we use $L_i =$ "$*$" to denote that the path P_i has no length constraint. We regard k_1 and k_2 as parameters, and study the parameterized complexity of the following problem.

Edge-Disjoint (L_1, L_2)-Paths
Instance: Graph $G = (V, E)$, two pairs (s_1, t_1) and (s_2, t_2) of vertices with $s_i \neq t_i$ for $i = 1, 2$.
Question: Does G contain (s_1, t_1)-paths P_1 and (s_1, t_1)-paths P_2 such that P_1 and P_2 share no edge and the length of P_i satisfies L_i?

☆ An earlier version was published in Proceedings of the 42nd International Workshop on Graph-Theoretic Concepts in Computer Science, Istanbul, Turkey, 2016, pp. 62–73.

E-mail addresses: lcai@cse.cuhk.edu.hk (L. Cai), junjie.ye@polyu.edu.hk (J. Ye).

1 Partially supported by GRF grant CUHK410212 of the Research Grants Council of Hong Kong.

2 Present address: Department of Computing, Hong Kong Polytechnic University, Hong Kong SAR, China.

https://doi.org/10.1016/j.tcs.2019.07.009
0304-3975/

Table 1
Running times of FPT algorithms for Edge-Disjoint (L_1, L_2)-Paths under 9 different length constraints, where $r = k_1 \log(k_1 + k_2) + k_2$.

	$\|P_2\| = *$	$\|P_2\| \geq k_2$	$\|P_2\| = k_2$	$\|P_2\| \leq k_2$
$\|P_1\| \leq k_1$	$2^{O(k_1 \log k_1)} m \log n$	$2^{O(r)} m \log n$	$O(5.24^{k_1+k_2} m \log^3 n)$	$O(2.01^{k_1+k_2} m \log n)$
$\|P_1\| = k_1$	$2^{O(k_1 \log k_1)} m \log^3 n$	$2^{O(r)} m \log^3 n$	$O(5.24^{k_1+k_2} m \log^3 n)$	
$\|P_1\| \geq k_1$	open	open		

For instance, Edge-Disjoint $(= k_1, *)$-Paths requires that $|P_1| = k_1$ but P_2 has no length constraint. In this paper, we focus on Edge-Disjoint (L_1, L_2)-Paths where at least one path has a length constraint. Without loss of generality, we may assume that P_1 always has a length constraint, i.e., $L_1 \in \{\leq k_1, = k_1, \geq k_1\}$. This gives us 9 different length constraints for Edge-Disjoint (L_1, L_2)-Paths, excluding symmetric cases.

Related Work. Edge-Disjoint (L_1, L_2)-Paths has been studied under the framework of classical complexity. Ohtsuki [21], Seymour [24], Shiloah [26], and Thomasssen [27] independently gave polynomial-time algorithms for Edge-Disjoint $(*, *)$-Paths. Tragoudas and Varol [28] proved the NP-completeness of Edge-Disjoint $(\leq k_1, \leq k_2)$-Paths, and Eilam-Tzoreff [12] showed the NP-completeness of Edge-Disjoint $(\leq k_1, *)$-Paths even when k_1 equals the (s_1, t_1)-distance. For Edge-Disjoint (L_1, L_2)-Paths with $L_1 \in \{= k_1, \geq k_1\}$ (same for $L_2 \in \{= k_1, \geq k_1\}$), we can easily establish their NP-completeness by reductions from the classical Hamiltonian Path problem.

As for the parameterized complexity, there are a few results in connection with our Edge-Disjoint (L_1, L_2)-Paths. Golovach and Thilikos [19] obtained an $2^{O(pl)} m \log n$-time algorithm for Edge-Disjoint Paths when every path has length at most l. For a single pair (s, t) of vertices, an (s, t)-path of length exactly l can be found in time $O(2.619^l m \log^2 n)$ [14, 25], $O^*(2.5961^l)$ [31] or $O^*(2.554^l)$ [29]. Note that the related problem of finding a path of length l can be solved in time $O(2.619^l n \log^2 n)$ [14,25]. For the problem of finding an (s, t)-path of length at least l, Bodlaender [3] derived an $O(2^{2l}(2l)!n + m)$-time algorithm; Gabow and Nie [17] designed an $l^l 2^{O(l)} mn \log n$-time algorithm; and FPT algorithms of Fomin et al. [14] for cycles and paths can be adapted to yield an $8^{l+o(l)} m \log^2 n$-time algorithm. Furthermore, Fomin et al. [15] obtained an $O^*(4.884^k)$-time algorithm. Recently, Araujo et al. [2] have studied vertex-disjoint (s, t)-paths with length constraints in digraphs.

Our Contributions. In this paper, we investigate the parameterized complexity of Edge-Disjoint (L_1, L_2)-Paths for the 9 different length constraints and obtain FPT algorithms for 7 of them (see Table 1 for a summary).

In particular, we use random partition in a nontrivial way to obtain FPT algorithms for Edge-Disjoint $(= k_1, *)$-Paths and Edge-Disjoint $(= k_1, \geq k_2)$-Paths. This is achieved by bounding the number of some special edges, called "nearby-edges", in the two paths P_1 and P_2 by a function of k_1 and k_2. We also consider polynomial kernels and prove that all 9 cases admit no polynomial kernel unless $NP \subseteq coNP/poly$, which easily extends to variations of edge/vertex-disjoint (L_1, L_2)-paths problems for undirected/directed graphs.

In the rest of the paper, we fix notation and give definitions in Section 2, present FPT algorithms for 7 cases in Section 3, and show the nonexistence of polynomial kernels in Section 4. We conclude with some open problems in Section 5.

2. Notation and definitions

Unless specified otherwise, all graphs $G = (V, E)$ in the paper are simple undirected connected graphs, and we use m and n, respectively, for the numbers of edges and vertices of G. For two vertices u and v in G, their distance is denoted by $d(u, v)$. We use $d(v, P)$ to denote the distance between a vertex v and a path P, i.e., $d(v, P) = \min_{u \in V(P)} d(v, u)$. For two vertices u, v on a path P, we use $P[u, v]$ to denote the (u, v)-section of P, i.e., (u, v)-path in P; and $P^{-1}[u, v]$ to denote $P[u, v]$ in reverse order. We refer to [30] for basic definitions of graph theory.

For simplicity, we write $O(2.01^{f(k)})$ for $2^{f(k)+o(f(k))}$ as the latter is $O((2 + \epsilon)^{f(k)})$ for any constant $\epsilon > 0$. In particular, $2^k k^{O(\log k)} = 2^{k+O(\log^2 k)} = O(2.01^k)$.

For an instance (I, k) of a parameterized problem Π, the main input I is encoded with some finite alphabet $\mathbf{\Sigma}$ and parameter $k \in \mathbf{N}$ is encoded in unary. Problem Π is *fixed-parameter tractable* (FPT in short) [11] if there is an algorithm that solves every instance (I, k) in time $f(k)|I|^{O(1)}$ for some computable function f. In this paper, kernels for Π refer to *generalized kernels* defined below.

Definition 1. (see [4]) A *generalized kernelization* from a parameterized problem Π to another parameterized problem Π' is an algorithm that takes time polynomial in $|I| + k$ for input $(I, k) \in \Pi$ and outputs an instance $(I', k') \in \Pi'$ such that
(a) (I, k) is a yes-instance of Π if and only if (I', k') is a yes-instance of Π', and
(b) both $|I'|$ and k' are bounded above by a computable function $g(k)$.
The output (I', k') is a *kernel*, and a *polynomial kernel* if $g(k)$ is a polynomial.

The above definition is naturally generalized to *polynomial compressions* by relaxing the target problem Π' to any problem (instead of parameterized problem), i.e., language $L \subseteq \mathbf{\Sigma}^*$.

Definition 2. (see [5]) Let Π be a parameterized problem and $L \subseteq \mathbf{\Sigma}^*$ a language. A *polynomial compression* from Π to L is an algorithm that takes time polynomial in $|I| + k$ for input $(I, k) \in \Pi$ and outputs a string $y \in \mathbf{\Sigma}^*$ such that
(a) (I, k) is a yes-instance of Π if and only if $y \in L$, and
(b) the length of y is bounded above by a polynomial of k.

For simplicity in discussions, we call a parameterized problem *incompressible* when it admits no polynomial compression (hence no polynomial kernel) under the assumption that NP $\not\subseteq$ coNP/poly.

For the purpose of derandomization, we need the following concepts of universal sets [1] and perfect hash functions [7]. A family of binary vectors of length l forms (l, s)-*universal sets* if for every subset of size s of the indices, all 2^s configurations appear in the family. A family of functions from $\{1, 2, \ldots, l\}$ to $\{1, 2, \ldots, d\}$ is an (l, d, s)-*perfect hash family* if for any subset $S \subseteq \{1, 2, \ldots, l\}$ of size s, there is a function in the family that is one-to-one on S. Here d is a power of 2 between $s(s-1)/2 + 2$ and $2s(s-1) + 4$.

3. FPT algorithms

One natural tool for finding edge-disjoint (L_1, L_2)-paths in a graph G is to use random partition: *Randomly partition edges of G to form two graphs G_1 and G_2, and then independently find, for each $i \in \{1, 2\}$, path P_i in G_i to satisfy length constraint L_i.*

This approach yields a randomized FPT algorithm when Edge-Disjoint (L_1, L_2)-Paths satisfies the following two conditions:

C1. Graph G admits a solution (P_1, P_2) such that the probability of "G_1 contains P_1 and G_2 contains P_2" is bounded below by a function of k_1 and k_2.
C2. It takes FPT time to find paths P_1 in G_1 and P_2 in G_2.

Indeed, straightforward applications of the above method yield FPT algorithms for Edge-Disjoint (L_1, L_2)-Paths when $L_i \in \{\le k_i, \; = k_i\}$ for $i \in \{1, 2\}$. Note that we can also obtain FPT algorithms for these three cases of (L_1, L_2) by using much involved representative sets based on Lemma 5.2 of Fomin et al. [14].

Theorem 1. Edge-Disjoint (L_1, L_2)-Paths *can be solved in* $O(2.01^{k_1+k_2} m \log n)$ *time for* $(L_1, L_2) = (\le k_1, \; \le k_2)$, *and* $O(5.24^{k_1+k_2} m \log^3 n)$ *time for* $(L_1, L_2) = (\le k_1, \; = k_2)$ *or* $(= k_1, \; = k_2)$.

Proof. For any solution (P_1, P_2) of Edge-Disjoint (L_1, L_2)-Paths and a random edge partition of G into two graphs G_1 and G_2, the probability that G_1 contains P_1 and G_2 contains P_2 is at least $1/2^{k_1+k_2}$ for all three cases of (L_1, L_2). Since it takes $O(m)$ time by BFS and $O(2.619^l m \log^2 n)$ time by an algorithm of Fomin et al. [14] to find an (s, t)-path of length at most l and exactly l, respectively, between two given vertices s and t, this random partition method gives us a randomized FPT algorithm with success probability at least $1/2^{k_1+k_2}$.

For derandomization, we use a family of (m, r)-universal sets, where $r = k_1 + k_2$, of size $2^r r^{O(\log r)} \log m$ [20]. Since

$$2^r r^{O(\log r)} \log m \cdot m = 2^{r + O(\log^2 r)} m \log n = O(2.01^r m \log n)$$

and

$$2^r r^{O(\log r)} \log m \cdot (2.619^{k_1} + 2.619^{k_2}) m \log^2 n = O(5.24^r m \log^3 n),$$

we obtain the claimed time bounds in the theorem. $\square$

For other cases of (L_1, L_2), a random edge-partition of G does not, unfortunately, guarantee condition **C1** because of possible long paths in a solution. To cope with such cases, we define some special edges called *nearby-edges* and then use random partition on such edges to ensure condition **C1** by limiting their numbers in some solutions by polynomials of k_1 and k_2.

Definition 3. A vertex v is a *nearby-vertex* if $\min\{d(v, s_1), d(v, t_1)\} \le k_1/2$, and an edge is a *nearby-edge* if its two end-vertices are both nearby-vertices.

In the next two subsections, we rely on random partition of nearby-edges to obtain FPT algorithms to solve Edge-Disjoint (L_1, L_2)-Paths for the following four cases of (L_1, L_2): $(\le k_1, *)$, $(= k_1, *)$, $(\le k_1, \ge k_2)$ and $(= k_1, \ge k_2)$.

We note that such a nontrivial way of applying random partition was initially used by Cai et al. [9] in two examples of their random separation method for graphs of bounded degeneracy, and was also used later by Cygan et al. [10] for Eulerian deletion.

3.1. One short and one unconstrained

To obtain FPT algorithms for Edge-Disjoint (L_1, L_2)-Paths when (L_1, L_2) is $(\leq k_1, *)$ or $(=k_1, *)$, we first give an upper bound on the number of nearby-edges in a special solution.

Lemma 1. *Let (s_1, t_1) and (s_2, t_2) be two pairs of vertices in a graph $G = (V, E)$, P_1 an (s_1, t_1)-path of length at most k_1, and P_2 a minimum-length (s_2, t_2)-path edge-disjoint from P_1. Then*

1. *all edges in P_1 are nearby-edges, and*
2. *P_2 contains at most $(k_1 + 1)^2$ nearby-vertices and $(k_1 + 1)^2 - 1$ nearby-edges.*

Proof. Statement 1 is obvious and we focus on Statement 2. For this purpose, we call a vertex a P_1*-near vertex* if its distance to P_1 is at most $k_1/2$, and show first that P_2 contains at most $(k_1 + 1)^2$ P_1-near vertices.

Consider an arbitrary vertex x in P_1 and define

$$N_x^* = \{v : v \text{ is a } P_1\text{-near vertex in } P_2 \text{ and } d(v, x) = d(v, P_1)\},$$

where $d(v, P_1)$ is the minimum distance between v and any vertex of P_1. In other words, for each $v \in N_x^*$, x is a vertex in P_1 closest to v.

Order vertices in N_x^* along P_2 from s_2 to t_2 and denote the first and last vertices by x_s and x_t respectively. In G, let P_s be a shortest (x_s, x)-path and P_t a shortest (x, x_t)-path. Then both P_s and P_t are edge-disjoint from P_1 as x is a vertex in P_1 closest to both x_s and x_t, and therefore $P_s P_t$ is an (x_s, x_t)-walk edge-disjoint from P_1. Note that $P_s P_t$ contains at most k_1 edges as both P_s and P_t have at most $k_1/2$ edges.

If the (x_s, x_t)-section of P_2 contains more than k_1 edges, then we can replace it by $P_s P_t$ to obtain an (s_2, t_2)-walk that is edge-disjoint from P_1 and shorter than P_2, contradicting the minimality of P_2. Therefore, the (x_s, x_t)-section of P_2 contains at most k_1 edges, implying that it contains at most $k_1 + 1$ P_1-near vertices, i.e., $|N_x^*| \leq k_1 + 1$. Since P_1 has at most $k_1 + 1$ vertices and every P_1-near vertex in P_2 belongs to N_x^* for some vertex x in P_1, we see that P_2 contains at most $(k_1 + 1)^2$ P_1-near vertices.

Since s_1 and t_1 are vertices of P_1, every nearby-vertex is a P_1-near vertex. Therefore P_2 contains at most $(k_1 + 1)^2$ nearby-vertices, and hence at most $(k_1 + 1)^2 - 1$ nearby-edges. □

The above corollary lays the ground for the following randomized FPT algorithm using random partition of nearby-edges to solve Edge-Disjoint $(\leq k_1, *)$-Paths. The algorithm also works for Edge-Disjoint $(= k_1, *)$-Paths by changing "length $\leq k_1$" to "length k_1" in Step 3.

Algorithm 1. Edge-Disjoint $(\leq k_1, *)$-Paths.

1. Find all nearby-edges by two rounds of BFS, one from s_1 and the other from t_1.
2. Randomly color each nearby-edge by color 1 or 2 with probability 1/2, and color all remaining edges of G by color 2. Let G_i $(i = 1, 2)$ be the graph consisting of edges of color i.
3. Find an (s_1, t_1)-path P_1 of length $\leq k_1$ in G_1, and an (s_2, t_2)-path P_2 in G_2. Return (P_1, P_2) as a solution if both P_1 and P_2 exist, and "No" otherwise.

We remark that for both problems in the following theorem, our derandomized version of Algorithm 1 actually finds a solution with minimum total length of the two paths whenever G admits a solution.

Theorem 2. Edge-Disjoint $(\leq k_1, *)$-Paths *can be solved in $2^{O(k_1 \log k_1)} m \log n$ time, and* Edge-Disjoint $(= k_1, *)$-Paths *can be solved in $2^{O(k_1 \log k_1)} m \log^3 n$ time.*

Proof. We focus on Edge-Disjoint $(\leq k_1, *)$-Paths as our analysis also works for Edge-Disjoint $(= k_1, *)$-Paths with one minor change. First we show that Algorithm 1 finds a solution in $O(m)$ time with probability $> 1/2^{k_1 + (k_1+1)^2}$ when G admits a solution, and then we derandomize the algorithm to obtain the claimed time bounds.

Let (P_1, P_2) be a solution of G that minimizes the length of P_2. By Lemma 1, we see that P_1 is entirely inside G_1 with probability $\geq 1/2^{k_1}$ and P_2 is entirely inside G_2 with probability $> 1/2^{(k_1+1)^2}$. Since it takes $O(m)$ time by BFS to find a shortest (s, t)-path between two vertices s and t, Algorithm 1 has probability $> 1/2^{k_1 + (k_1+1)^2}$ to find a solution in $O(m)$ time.

We now discuss derandomization of Algorithm 1. Let m' be the number of nearby-edges and $r = k_1 + (k_1 + 1)^2$. We can use standard (m', r)-universal sets to derandomize it and obtain a deterministic FPT algorithm with running time

$$2^r r^{O(\log r)} \log n \cdot m' = O(2.01^{k_1^2} m \log n).$$

We can reduce $O(2.01^{k_1^2})$ in running time to $2^{O(k_1 \log k_1)}$ by using an (m', d, r)-perfect hash family for derandomization, where d is a power of 2 between $r(r-1)/2+2$ and $2r(r-1)+4$. Note that such a number exists as $2r(r-1)+4 \geq 2(r(r-1)/2+2)$. Bshouty [7] has shown that an (m', d, r)-perfect hash family of size

$$O\left(\frac{d^2r^2 \log m'}{(d - r(r-1)/2 - 1)^2}\right) = O(r^6 \log m')$$

can be constructed in linear time. In Step 2 of Algorithm 1, we want to separate P_1 from P_2 by making P_1 color 1 and P_2 color 2. Instead of random colorings, we try each pair (f, F), where f is a function in an (m', d, r)-perfect hash family and F is a subset of $\{1, 2, \ldots, d\}$ with $|F| = k_1$. We first identify the set of nearby-edges with $\{1, 2, \ldots, m'\}$. Given a particular pair (f, F), we color a nearby-edge e by color 1 if $f(e) \in F$, and color all other edges color 2. By the definition of perfect hash family, if there is a solution (P_1, P_2), there will be a function f that is one-to-one on the set of nearby-edges in $E(P_1) \cup E(P_2)$ and a subset F such that $f(e) \in F$ if $e \in E(P_1)$ and $f(e) \notin F$ if e is a nearby-edge in $E(P_2)$. Since an (m', d, r)-perfect hash family has size $O(r^6 \log m')$ and can be constructed in linear time, there are

$$O(r^6 \log m') \cdot \binom{d}{k_1} = 2^{O(k_1 \log k_1)} \log m'$$

choices for pairs (f, F). It follows that the total running time for the deterministic algorithm is

$$2^{O(k_1 \log k_1)} \log m' \cdot m = 2^{O(k_1 \log k_1)} m \log n.$$

For Edge-Disjoint $(=k_1, *)$-Paths, Step 3 takes more time as it takes time $O(2.619^{k_1} m \log^2 n)$ to find an (s_1, t_1)-path P_1 of length k_1. Therefore our deterministic FPT algorithm for Edge-Disjoint $(=k_1, *)$-Paths runs in time

$$2^{O(k_1 \log k_1)} \log m' \cdot 2.619^{k_1} m \log^2 n = 2^{O(k_1 \log k_1)} m \log^3 n. \quad \square$$

3.2. One short and one long

We now consider Edge-Disjoint (L_1, L_2)-Paths when (L_1, L_2) is $(\leq k_1, \geq k_2)$ or $(=k_1, \geq k_2)$. These two cases are more complicated because of length lower bound on (s_2, t_2)-paths. Fortunately, we can still put a good bound on the number of nearby-edges in some special solutions, which enables us to use random partition on nearby-edges to obtain FPT algorithms for these two cases as well. The proof of the following lemma is more involved than that of Lemma 1.

Lemma 2. *Let (s_1, t_1) and (s_2, t_2) be two pairs of vertices in a graph $G = (V, E)$, P_1 an (s_1, t_1)-path of length at most k_1, and P_2 a minimum-length (s_2, t_2)-path that is edge-disjoint from P_1 and has length at least k_2. Then*

1. *all edges in P_1 are nearby-edges, and*
2. *P_2 contains at most $k_1^2 + 3k_1 + 2k_2 + 3$ nearby-vertices and $k_1^2 + 3k_1 + 2k_2 + 2$ nearby-edges.*

Proof. Statement (1) is obvious by definition and we focus on Statement (2). For path P_2, let s^* be its (k_2+1)-th vertex and we use s^* to divide P_2 into $P_2^s = P_2[s_2, s^*]$ and $P_2^t = P_2[s^*, t_2]$. Obviously P_2^s can have at most $k_2 + 1$ nearby-vertices as it has $k_2 + 1$ vertices only. For nearby-vertices in P_2^t, we arrange them into two groups and then determine the size of each group separately.

Consider an arbitrary nearby-vertex v. By definition, v has a path Q of length at most $k_1/2$ to s_1 or t_1. Let v^* be the first vertex in P_1 or P_2^s when we travel along Q from v. Since s_1 and t_1 are vertices of P_1, v^* always exists and any such v^* is called a *docking vertex* of v. We call v a *near-P_1 vertex* (resp., *near-P_2^s vertex*) if it has a docking vertex in P_1 (resp., P_2^s). Therefore every nearby-vertex is either near-P_1, near-P_2^s, or both.

We also call the (v, v^*)-section $Q[v, v^*]$ of Q a *docking path*. It is important to note that a docking path $Q[v, v^*]$ has length at most $k_1/2$ and $Q[v, v^*] \backslash v^*$ is always vertex-disjoint from both P_1 and P_2^s.

We are ready to put a bound on the number of near-P_1 vertices in P_2^t. For this purpose, we define for each vertex $x \in V(P_1) \backslash V(P_2^s)$ the following set of near-P_1 vertices:

$$D(x) = \{v : v \text{ is a near-}P_1 \text{ vertex in } P_2^t \text{ and } x \text{ is a docking vertex of } v.\}$$

Following the same arguments for N_x^* in the proof of Lemma 1, we can use docking paths $Q[x_s, x]$ and $Q^{-1}[x_t, x]$, respectively, as paths P_s and P_t in that proof to show that $|D(x)| \leq k_1 + 1$. Therefore P_2^t contains at most $(k_1+1)^2$ near-P_1 vertices as $|V(P_1) \backslash V(P_2^s)| \leq k_1 + 1$.

Next we consider the number of near-P_2^s vertices in P_2^t. Suppose that P_2^t contains at least $k_1 + k_2 + 2$ near-P_2^s vertices. Let y be the $(\lceil k_1/2 \rceil + 2)$-th near-$P_2^s$ vertex in P_2^t. Then there is a docking path Q from some docking vertex y' of y in P_2^s to vertex y. Let z be the last vertex of P_2^t that also appears in Q, and note that z lies in $P_2[y, t_2]$. Denote the (k_2+1)-th

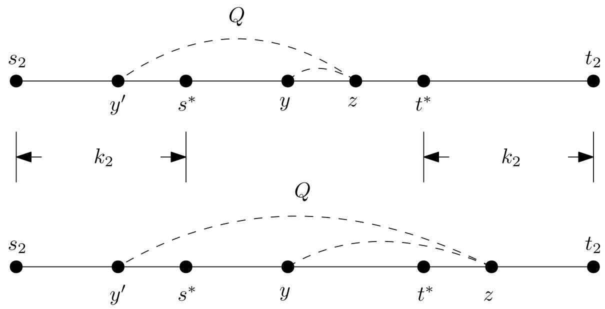

Fig. 1. Two possible cases for the intersection of Q and P_2^t.

last vertex of P_2 by t^*, and we consider two cases. For convenience, we call an (s_2, t_2)-path a *valid* (s_2, t_2)*-path* if it is edge-disjoint from P_1 and has length at least k_2.

Case 1. Vertex z is in $P_2[y, t^*]$ (see the top part of Fig. 1).

Since Q is edge-disjoint from P_1 and vertex-disjoint from $P_2^s \backslash y'$, we can obtain from P_2 an (s_2, t_2)-path P by replacing $P_2[y', z]$ with $Q[y', z]$. Clearly $|P|$ is a valid (s_2, t_2)-path as it contains $P_2[t^*, t_2]$ which has length k. However, since $|P_2[s^*, y]| \geq \lceil k_1/2 \rceil + 1$ by the definition of y, we see that $|P_2[y', z]| > |Q[y', z]|$ as $|Q| \leq k_1/2$ and therefore $|P| < |P_2|$, which is impossible by the minimality of P_2.

Case 2. Vertex z is in $P_2[t^*, t_2]$ (see the bottom part of Fig. 1).

Since Q is edge-disjoint from P_1, we can obtain from P_2 an (s_2, t_2)-walk W edge-disjoint from P_1 by replacing $P_2[y, z]$ with $Q^{-1}[z, y]$, which implies a valid (s_2, t_2)-path P as the first $k_2 + 1$ vertices of W are exactly vertices of P_2^s. However $|Q[z, y]| < k_1/2$ and $|P_2[y, z]| \geq k_1/2$, and hence $|P| \leq |W| < |P_2|$, which is again impossible by the minimality of P_2.

Since both cases lead to a contradiction to the minimality of P_2, we see that P_2^t can contain at most $k_1 + k_2 + 1$ near-P_2^s vertices. Together with at most $(k_1 + 1)^2$ near-P_1 vertices in P_2^t and $k_2 + 1$ vertices in P_2^s, we conclude that P_2 contains at most $k_1^2 + 3k_1 + 2k_2 + 3$ nearby-vertices, and hence at most $k_1^2 + 3k_1 + 2k_2 + 2$ nearby-edges. $\square$

The above corollary enables us to obtain a randomized FPT for Edge-Disjoint ($\leq k_1$, $\geq k_2$) by replacing Step 3 of Algorithm 1 as follows:

Step 3: Find an (s_1, t_1)-path P_1 of length $\leq k_1$ in G_1, and an (s_2, t_2)-path P_2 of length $\geq k_2$ in G_2. Return (P_1, P_2) as a solution if both P_1 and P_2 exist, and "No" otherwise.

We remark that for both problems in the following theorem, our derandomized algorithm actually finds a solution with minimum total length of the two paths whenever G admits a solution.

Theorem 3. *Both* Edge-Disjoint ($\leq k_1, \geq k_2$)-Paths *and* Edge-Disjoint ($= k_1, \geq k_2$)-Paths *can be solved in* $2^{O(k_1 \log(k_1+k_2)+k_2)} m \log^3 n$ *time.*

Proof. We focus on Edge-Disjoint ($\leq k_1$, $\geq k_2$)-Paths as our analysis also works for Edge-Disjoint ($= k_1$, $\geq k_2$)-Paths with one minor change. Let (P_1, P_2) be a solution of G that minimizes the length of P_2. By Lemma 2, we see that P_1 is entirely inside G_1 with probability $\geq 1/2^{k_1}$ and P_2 is entirely inside G_2 with probability $\geq 1/2^{k_1^2+3k_1+2k_2+2}$. Since an (s_2, t_2)-path P_2 of length $\geq k_2$ in G_2 can be found in time $8^{k_2+o(k_2)} m \log^2 n$ [14], our randomized algorithm runs in the same amount of time with success probability $\geq 1/2^{k_1^2+4k_1+2k_2+2}$.

For derandomization, let m' be the number of nearby-edges of G and set $r = k_1^2 + 4k_1 + 2k_2 + 2$. Let d be a power of 2 between $r(r-1)/2 + 2$ and $2r(r-1) + 4$. Similarly to Algorithm 1, we use an (m', d, r)-perfect hash family to derandomize our algorithm and obtain a deterministic FPT algorithm for Edge-Disjoint ($\leq k_1$, $\geq k_2$)-Paths with running time

$$O(r^6 \log m') \cdot \binom{d}{k_1} \cdot 8^{k_2+o(k_2)} m \log^2 n = 2^{O(k_1 \log(k_1+k_2)+k_2)} m \log^3 n.$$

For Edge-Disjoint ($= k_1$, $\geq k_2$)-Paths, Step 3 finds in G_1 an (s_1, t_1)-path P_1 of length k_1 (instead of length $\leq k_1$) in $O(2.619^{k_1} m \log^2 n)$ time [14]. Therefore we obtain a deterministic FPT algorithm for the problem with running time

$$O(r^6 \log m') \cdot \binom{d}{k_1} \cdot O(2.619^{k_1} m \log^2 n + 8^{k_2+o(k_2)} m \log^2 n)$$
$$= 2^{O(k_1 \log(k_1+k_2)+k_2)} m \log^3 n. \quad \square$$

4. Incompressibility of disjoint-paths problems

Having obtained FPT algorithms to solve seven Edge-Disjoint (L_1, L_2)-Paths problems, we show in this section the nonexistence of polynomial kernels for Edge-Disjoint (L_1, L_2)-Paths.

Theorem 4. *For each of the nine different length constraints* (L_1, L_2), Edge-Disjoint (L_1, L_2)-Paths *admits no polynomial compression (hence no polynomial kernel) unless* $NP \subseteq coNP/poly$, *even when the two terminal pairs are identical.*

Remark. The above theorem also holds for digraphs, and for corresponding vertex-disjoint versions on both undirected graphs and digraphs, which can be shown easily by standard reductions for undirected/directed graphs.

4.1. Tools for incompressibility

Our tools for incompressibility are *polynomial parameter transformation* (*ppt-reduction* in short) and *relaxed-composition*.

Definition 4. (see [5,6]) A *ppt-reduction* from a parameterized problem Π to another parameterized problem Π' is an algorithm that, for input $(I, k) \in \Pi$, takes time polynomial in $|I| + k$ and outputs an instance $(I', k') \in \Pi'$ such that
(a) (I, k) is a yes-instance of Π if and only if (I', k') is a yes-instance of Π', and
(b) parameter k' is bounded by a polynomial of k.

Theorem 5. (see [5]) *If there is a ppt-reduction from a parameterized problem* Π *to another parameterized problem* Π', *then* Π' *admits no polynomial compression (hence no polynomial kernel) whenever* Π *admits no polynomial compression.*

Relaxed-composition algorithms were defined by Cai and Cai [8] to form a relaxation of composition algorithms introduced by Bodlaender et al. [4] in their pioneer work on the nonexistence of polynomial kernels, and a clipped version of cross-composition [5] without polynomial equivalence relations.

Definition 5. (see [8]) A relaxed-composition algorithm for a parameterized problem Π takes p instances $(I_1, k), \dots, (I_p, k) \in \Pi$ as input and, in time polynomial in $\sum_{i=1}^{p} |I_i| + k$, outputs an instance $(I', k') \in \Pi$ such that
(a) (I', k') is a yes-instance of Π if and only if some (I_i, k) is a yes-instance of Π, and
(b) k' is polynomial in $\max_i^p |I_i| + \log p$.

Note that relaxed-composition algorithms relax the requirement in composition algorithms [4] for parameter k' from polynomial in k to polynomial in $\max_{i=1}^{p} |I_i| + \log p$. As observed by Cai and Cai [8], Bodlaender et al. [4], together with a result of Fortnow and Santhanam [16], implicitly proved the following theorem.

Theorem 6. (see [4,5,16]) *If an NP-complete parameterized problem admits a relaxed-composition algorithm, then it has no polynomial compression (hence no polynomial kernel), unless* NP $\subseteq$ coNP/poly.

4.2. One long or of exact length

We start with incompressibility of Edge-Disjoint (L_1, L_2)-Paths when at least one path is long or of exact length, i.e., when the length constraints (L_1, L_2) are $(\le k_1, = k_2)$, $(\le k_1, \ge k_2)$, $(= k_1, = k_2)$, $(= k_1, \ge k_2)$, $(= k_1, *)$, $(\ge k_1, \ge k_2)$, or $(\ge k_1, *)$.

First we show that the following two path problems are incompressible by relaxed-compositions, and then give simple ppt-reductions from these two problems to our problems under the above seven length constraints (L_1, L_2).

Long Path: For two given vertices s and t in a graph G, does G contain an (s, t)-path of length at least k?

Exact-Length Path: For two given vertices s and t in a graph G, does G contain an (s, t)-path of length exactly k?

Note that Long Path and Exact-Length Path are both NP-complete by a simple reduction from the classical Hamiltonian Path problem ([GT39] in [18]).

Lemma 3. *Neither* Exact-Length Path *nor* Long Path *admits polynomial compression unless* $NP \subseteq coNP/poly$.

Proof. For a collection of graphs with terminals s_i and t_i for the i-th graph G_i, we construct a graph G' by merging all s_i into one new terminal s and all t_i into one new terminal t. Clearly, G' contains an (s, t)-path of length k (resp., $\ge k$) if and only if one of G_i contains an (s_i, t_i)-path of length k (resp., $\ge k$). By Theorem 6, this relaxed-composition establishes the lemma. $\square$

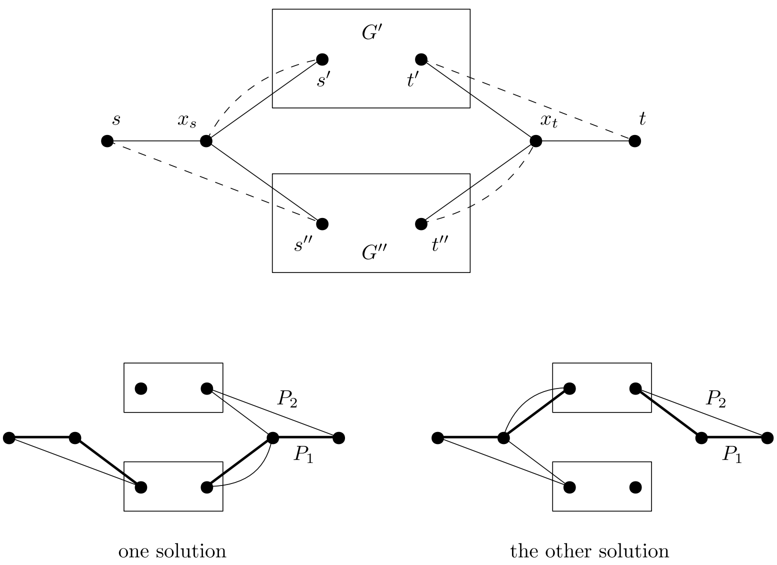

Fig. 2. Construction of graph G for relaxed-composition of two instances, where each dashed line indicates a path of length $k_1 + 4$. Graph G has exactly two possible solutions as shown in the figure.

The above lemma enables us to use the following straightforward ppt-reduction from either Exact-Length Path or Long Path to establish the incompressibility of the seven cases of length constraints for Edge-Disjoint (L_1, L_2)-Paths:

> For a graph G with two vertices s and t, we add a path on $l - 1$ new vertices from s to t to form a new graph G' with two identical terminal pairs (s, t).

It is obvious that G contains an (s, t)-path of length k if and only if G' contains two edge-disjoint (s, t)-paths one of length l and the other length k. Therefore we can set l to either k_1 or k_2 and use the above construction as a ppt-reduction from Exact-Length Path or Long Path to settle seven cases in Theorem 4.

4.3. Two short paths

Now we consider the remaining two cases of length constraints $(\leq k_1, \leq k_2)$ and $(\leq k_1, *)$. The ppt-reductions for the other seven cases do not work as finding an (s, t)-path of length at most k is polynomial time solvable. Here we will give relaxed-composition algorithms to establish the incompressibility of Edge-Disjoint (L_1, L_2)-Paths for these two cases.

Lemma 4. Edge-Disjoint $(\leq k_1, \leq k_2)$-Paths *admits no polynomial compression (hence no polynomial kernel) unless* NP $\subseteq$ coNP/poly*, even when two terminal pairs are identical.*

Proof. Let $\mathcal{I}$ be a collection of p instances each with the same parameters k_1, k_2. We will construct a relaxed-composition of $\mathcal{I}$ to establish the lemma. For this purpose, we first consider two arbitrary instances $I' = (G', k_1, k_2, (s', t'))$ and $I'' = (G'', k_1, k_2, (s'', t''))$ of the problem with identical terminal pairs, and construct from them an instance, denoted $I' \oplus I''$, such that $I' \oplus I''$ is a yes-instance if and only if one of I' and I'' is a yes-instance.

For instance $I' \oplus I''$, we construct a graph G with two identical terminal pairs (s, t), and set parameters of $I' \oplus I''$ to $k_1 + 4, k_2 + 3(k_1 + 4) + 1$ as follows (see Fig. 2).

1. Take graphs G' and G'', add vertices x_s and x_t, and a terminal pair (s, t).
2. Add edge sx_s, a path of length $k_1 + 4$ connecting s and s'', edges $x_s s'$ and $x_s s''$, and a path of length $k_1 + 4$ connecting x_s and s'.
3. Add edge tx_t, a path of length $k_1 + 4$ connecting t and t', edges $x_t t'$ and $x_t t''$, and a path of length $k_1 + 4$ connecting x_t and t''.

To see that $I' \oplus I''$ satisfies the required property, we consider possible solutions of (P_1, P_2) in G. As shown in Fig. 2, P_1 can be formed in exactly two different ways, and each forces a unique P_2. Therefore there are exactly two possible solutions (P_1, P_2) for G and it is easily checked that they have required lengths if and only if their sections inside G' (resp., G'') are bounded above by k_1 and k_2.

With the construction of $I' \oplus I''$ in hand, we can easily use the following divide-and-conquer Algorithm $RC(\mathcal{I})$ to compute a relaxed-composition of $\mathcal{I}$. We may assume that $|\mathcal{I}| = 2^d$ for some integer d, as we can always add some dummy no-instances to $\mathcal{I}$.

Algorithm $RC(\mathcal{I})$.
Input: A collection $\mathcal{I}$ of 2^d instances all having the same parameter values.
Output: A relaxed-composition of $RC(\mathcal{I})$.

If $\mathcal{I}$ contains two instances I' and I'' only
then return $I' \oplus I''$
else evenly split $\mathcal{I}$ into $\{\mathcal{I}', \mathcal{I}''\}$ and **return** $RC(\mathcal{I}') \oplus RC(\mathcal{I}'')$.

Since $|\mathcal{I}'| = |\mathcal{I}''| = 2^{d-1}$, $RC(\mathcal{I}')$ and $RC(\mathcal{I}'')$ have the same parameter values. Therefore Algorithm $RC(\mathcal{I})$ correctly returns an instance that is a yes-instance if and only if at least one instance in $\mathcal{I}$ is a yes-instance.

Let $k_1^{(d)}, k_2^{(d)}$ be the two parameters of $RC(\mathcal{I})$. Then we have $k_1^{(0)} = k_1$, $k_2^{(0)} = k_2$, and

$$\begin{cases} k_1^{(d)} = k_1^{(d-1)} + 4 \\ k_2^{(d)} = k_2^{(d-1)} + 3(k_1^{(d-1)} + 4) + 1. \end{cases}$$

This yields $k_1^{(d)} = k_1 + 4d$ and $k_2^{(d)} = k_2 + 3dk_1 + d(6d + 7)$.

Note that both parameters are upper bounded by a polynomial in $n + \log p$ as $d = \log p$ and $k_1, k_2 \le n$. Also the construction of $I' \oplus I''$ takes time linear in $|I'| + |I''|$, and hence the algorithm constructs a relaxed-composition of $\mathcal{I}$ in time linear in the total length of instances in $\mathcal{I}$. Since the problem Edge-Disjoint $(\le k_1, \le k_2)$-Paths is NP-complete [28], it follows from Theorem 6 that the problem admits no polynomial compression (hence no polynomial kernel) unless $NP \subseteq coNP/poly$. $\square$

The proof of the above lemma also works for Edge-Disjoint $(\le k_1, *)$-Paths by discarding the second parameter, and therefore the problem admits no polynomial compression (hence no polynomial kernel) unless NP $\subseteq$ coNP/poly.

5. Concluding remarks

We have obtained FPT algorithms to solve Edge-Disjoint (L_1, L_2)-Paths for seven of the nine different cases of length constraints (L_1, L_2). On the other hand, we have also established the nonexistence of polynomial kernels for all nine cases, which also easily extends to variations of edge/vertex-disjoint (L_1, L_2)-paths problems for undirected/directed graphs.

There are still many interesting problems in connection with the work of this paper, and here we highlight a few of them.

Problem 1. Determine parameterized complexities of Edge-Disjoint $(\ge k_1, *)$-Paths and Edge-Disjoint $(\ge k_1, \ge k_2)$-Paths.

Since Edge-Disjoint $(\ge k_1, *)$-Paths is equivalent to Edge-Disjoint $(\ge k_1, \ge k_2)$-Paths for $k_2 = 1$, an FPT algorithm for the latter problem is also an FPT algorithm for the former one.

We may also consider edge-disjoint paths when solution paths (P_1, P_2) need to satisfy additional properties, and the following problem is related to vertex-disjoint variation.

Problem 2. Determine the parameterized complexity of Edge-Disjoint $(\le k_1, \le k_2)$-Paths when we also want to minimize the number of vertices shared by solution paths (P_1, P_2).

Of course, we can consider edge-disjoint paths with length constraints for digraphs, which appear to be harder than these problems on undirected graphs. Note that it is straightforward to obtain FPT algorithms by random partition for (L_1, L_2) being $(\le k_1, \le k_2), (= k_1, \le k_2)$ or $(= k_1, = k_2)$, but structural properties similar to Lemma 1 or Lemma 2 seem unlikely for digraphs.

Problem 3. For digraphs, determine the parameterized complexity of Edge-Disjoint (L_1, L_2)-Paths.

Finally, we can also study vertex-disjoint paths problems with length constraints for both undirected and directed graphs.

Problem 4. Determine the parameterized complexity of Vertex-Disjoint (L_1, L_2)-Paths for undirected/directed graphs.

Declaration of Competing Interest

None declared.

Acknowledgements

The authors are grateful to the reviewers for their constructive suggestions.

References

[1] N. Alon, R. Yuster, U. Zwick, Color-coding, J. ACM 42 (4) (1995) 844–856.
[2] J. Araújo, V.A. Campos, A.K. Maia, I. Sau, A. Silva, On the complexity of finding internally vertex-disjoint long directed paths, in: Latin American Symposium on Theoretical Informatics, Springer, 2018, pp. 66–79.
[3] H.L. Bodlaender, On linear time minor tests with depth-first search, J. Algorithms 14 (1) (1993) 1–23.
[4] H.L. Bodlaender, R.G. Downey, M.R. Fellows, D. Hermelin, On problems without polynomial kernels, J. Comput. Syst. Sci. 75 (8) (2009) 423–434.
[5] H.L. Bodlaender, B.M. Jansen, S. Kratsch, Kernelization lower bounds by cross-composition, SIAM J. Discrete Math. 28 (1) (2014) 277–305.
[6] H.L. Bodlaender, S. Thomassé, A. Yeo, Kernel bounds for disjoint cycles and disjoint paths, Theor. Comput. Sci. 412 (35) (2011) 4570–4578.
[7] N.H. Bshouty, Linear time constructions of some d-restriction problems, in: Proceedings of the 9th International Conference on Algorithms and Complexity, Springer, 2015, pp. 74–88.
[8] L. Cai, Y. Cai, Incompressibility of H-free edge modification problems, Algorithmica 71 (3) (2014) 731–757.
[9] L. Cai, S.M. Chan, S.O. Chan, Random separation: a new method for solving fixed-cardinality optimization problems, in: Proceedings of the 2nd International Workshop on Parameterized and Exact Computation, in: Lecture Notes in Computer Science, vol. 4169, Springer, 2006, pp. 239–250.
[10] M. Cygan, D. Marx, M. Pilipczuk, M. Pilipczuk, I. Schlotter, Parameterized complexity of Eulerian deletion problems, Algorithmica 68 (1) (2014) 41–61.
[11] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, New York, 1999.
[12] T. Eilam-Tzoreff, The disjoint shortest paths problem, Discrete Appl. Math. 85 (2) (1998) 113–138.
[13] S. Even, A. Itai, A. Shamir, On the complexity of timetable and multicommodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.
[14] F.V. Fomin, D. Lokshtanov, F. Panolan, S. Saurabh, Efficient computation of representative families with applications in parameterized and exact algorithms, J. ACM 63 (4) (2016) 29.
[15] F.V. Fomin, D. Lokshtanov, F. Panolan, S. Saurabh, M. Zehavi, Long directed (s,t)-path: FPT algorithm, Inf. Process. Lett. (2018).
[16] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, J. Comput. Syst. Sci. 77 (1) (2011) 91–106.
[17] H.N. Gabow, S. Nie, Finding long paths, cycles and circuits, in: Algorithms and Computation, Springer, 2008, pp. 752–763.
[18] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
[19] P.A. Golovach, D.M. Thilikos, Paths of bounded length and their cuts: parameterized complexity and algorithms, Discrete Optim. 8 (1) (2011) 72–86.
[20] M. Naor, L.J. Schulman, A. Srinivasan, Splitters and near-optimal derandomization, in: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, IEEE, 1995, pp. 182–191.
[21] T. Ohtsuki, The two disjoint path problem and wire routing design, in: Proceedings of the 17th Symposium of Research Institute of Electric Communication on Graph Theory and Algorithms, Springer-Verlag, 1980, pp. 207–216.
[22] J.B. Orlin, Max flows in O(nm) time, or better, in: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, ACM, 2013, pp. 765–774.
[23] N. Robertson, P.D. Seymour, Graph minors. XIII. The disjoint paths problem, J. Comb. Theory, Ser. B 63 (1) (1995) 65–110.
[24] P.D. Seymour, Disjoint paths in graphs, Discrete Math. 29 (3) (1980) 293–309.
[25] H. Shachnai, M. Zehavi, Representative families: a unified tradeoff-based approach, J. Comput. Syst. Sci. 82 (3) (2016) 488–502.
[26] Y. Shiloach, A polynomial solution to the undirected two paths problem, J. ACM 27 (3) (1980) 445–456.
[27] C. Thomassen, 2-linked graphs, Eur. J. Comb. 1 (4) (1980) 371–378.
[28] S. Tragoudas, Y.L. Varol, Computing disjoint paths with length constraints, in: Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 1997, pp. 375–389.
[29] D. Tsur, Faster deterministic parameterized algorithm for k-path, arXiv preprint arXiv:1808.04185, 2018.
[30] D.B. West, Introduction to Graph Theory, Prentice Hall, Upper Saddle River, 2001.
[31] M. Zehavi, Mixing color coding-related techniques, in: Proceedings of the 23rd Annual European Symposium on Algorithms, Springer, 2015, pp. 1037–1049.