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

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. [5] have shown that the vertex-disjoint variation of Edge-Disjoint Paths admits no polynomial kernel unless \( NP \subseteq \text{coNP}/\text{poly} \).

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 \( L_i \in \{ \leq k_i, = k_i, \geq k_i, \ast \} \) indicates that the length of \( P_i \) needs to satisfy \( L_i \). In particular, we use \( L_i = \ast \) 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.

\textbf{EDGE-DISJOINT} \((L_1, L_2)\)-\textbf{PATHS}

\textbf{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 \( i = 1, 2 \).

\textbf{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 \( L_i \)?

---


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

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

Partial support by GRF grant CUHK410212 of the Research Grants Council of Hong Kong.

© 2019 Published by Elsevier B.V.
For instance, \textsc{Edge-Disjoint} (= $k_1$, *)-Paths requires that $|P_1| = k_1$ but $P_2$ has no length constraint. In this paper, we focus on \textsc{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 \textsc{Edge-Disjoint} ($L_1$, $L_2$)-Paths, excluding symmetric cases.

**Related Work.** \textsc{Edge-Disjoint} ($L_1$, $L_2$)-Paths has been studied under the framework of classical complexity. Ohtsuki [21], Seymour [24], Shiloah [26], and Thomassen [27] independently gave polynomial-time algorithms for \textsc{Edge-Disjoint} (*, *)-Paths. Tragoudas and Varol [28] proved the NP-completeness of \textsc{Edge-Disjoint} ($\leq k_1, \leq k_2$)-Paths, and Elam-Tzoref [12] showed the NP-completeness of \textsc{Edge-Disjoint} ($\leq k_1, *$)-Paths even when $k_1$ equals the $(s_1, t_1)$-distance. For \textsc{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 \textsc{Hamiltonian Path} problem.

As for the parameterized complexity, there are a few results in connection with our \textsc{Edge-Disjoint} ($L_1$, $L_2$)-Paths. Golovach and Thilikos [19] obtained an \textit{O}$^*(pl)m\log n$-time algorithm for \textsc{Edge-Disjoint} Paths when every path has length at most $l$. For a single pair of vertices, an $(s, t)$-path of length exactly $l$ can be found in time $O((2.619m\log^2n)$ [14, 25], $O^*(2.596l)$ [31] or $O^*(2.554l)$ [29]. Note that the related problem of finding a path of length $l$ can be solved in time $O(2.619m\log^2n)$ [14, 25]. For the problem of finding an $(s, t)$-path of length at least $l$, Bodlaender [3] derived an $O(2^l(2l/n + m))$-time algorithm; Gabow and Nie [17] designed an $\textit{O}(l^2|\Sigma|^2|\Pi|)$-time algorithm; and FPT algorithms of Fomin et al. [14] for cycles and paths can be adapted to yield an $\textit{O}(l^3|\Sigma|^2|\Pi|)$-time algorithm. Furthermore, Fomin et al. [15] obtained an $O^*(4.884^l)$-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 \textsc{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 \textsc{Edge-Disjoint} (= $k_1$, *)-Paths and \textsc{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., $d(v, P) = \min_{v \in V(P)} d(v, u)$. 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 $2^{f(k) + \alpha(f(k))}$ as the latter is $O((2 + \epsilon)^{f(k)})$ for any constant $\epsilon > 0$. In particular, $2^{k^2O(k)} = 2^{k^2+O(k^2)} = O(2.01^{|k|})$.

For an instance $(I, k)$ of a parameterized problem $\Pi$, the main input $I$ is encoded with some finite alphabet $\Sigma$ and parameter $k \in \mathbb{N}$ is encoded in unary. Problem $\Pi$ is \textit{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 $\Pi$ refer to generalized kernels defined below.

**Definition 1.** (see [4]) A \textit{generalized kernelization} from a parameterized problem $\Pi$ to another parameterized problem $\Pi'$ 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 $\Pi$ if and only if $(I', k')$ is a yes-instance of $\Pi'$, and

(b) both $|I'|$ and $k'$ are bounded above by a computable function $g(k)$.

The output $(I', k')$ is a \textit{kernel}, and a \textit{polynomial kernel} if $g(k)$ is a polynomial.

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

| $|P_1| < k_1$ | $|P_1| = k_1$ | $|P_1| \geq k_1$ |
|----------------|----------------|----------------|
| $|P_2| = k_1$ | $2^{O(|P_2| \log |P_2| \log n)}$ | $2^{O(|P_2| \log n)}$ |
| $|P_2| \geq k_2$ | $2^{O(|P_2| \log n)}$ | $0(2.01^{|P_2| \log |P_2| \log n})$ |
| $|P_2| = k_2$ | $2^{O(|P_2| \log n)}$ | $0(2.01^{|P_2| \log |P_2| \log n})$ |

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


Definition 2. (see [5]) Let $\Pi$ be a parameterized problem and $L \subseteq \Sigma^*$ a language. A polynomial compression from $\Pi$ to $L$ is an algorithm that takes time polynomial in $|l| + k$ for input $(l, k) \in \Pi$ and outputs a string $y \in \Sigma^*$ such that

(a) $(l, k)$ is a yes-instance of $\Pi$ 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 $\text{NP} \not\subset \text{coNP}/\text{poly}$.

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 $2^l$ 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 $\text{EDGE-DISJOINT} (L_1, L_2)$-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 $\text{EDGE-DISJOINT} (L_1, L_2)$-PATHS when $L_i \in \{k_i, k_i\}$ 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. $\text{EDGE-DISJOINT} (L_1, L_2)$-PATHS can be solved in $O(2.01^{k_1+k_2}m \log n)$ time for $(L_1, L_2) = (\leq k_1, \leq k_2)$, and $O(5.24^{k_1+k_2}m \log^3 n)$ time for $(L_1, L_2) = (\leq k_1, = k_2)$ or $(= k_1, = k_2)$.

Proof. For any solution $(P_1, P_2)$ of $\text{EDGE-DISJOINT} (L_1, L_2)$-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 $O(2.619m \log^2 n)$ 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. $\Box$

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 $\min\{d(v, s_1), d(v, t_1)\} \leq k_1/2$, and an edge is a nearby-edge if its two endpoints are both nearby-vertices.

In the next two subsections, we rely on random partition of nearby-edges to obtain FPT algorithms to solve $\text{EDGE-DISJOINT} (L_1, L_2)$-PATHS for the following four cases of $(L_1, L_2)$: $(\leq k_1, *)$, $(= k_1, *)$, $(\leq k_1, \geq k_2)$ and $(= k_1, \geq k_2)$.

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 \textsc{Edge-Disjoint} \((L_1, L_2)\)-\textsc{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 \(P_sP_t\) is an \((x_s, x_t)\)-walk edge-disjoint from \(P_1\). Note that \(P_sP_t\) contains at most \(k_1\) edges as \(P_s\) 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 \(P_sP_t\) 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. \(\square\)

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

**Algorithm 1.** \textsc{Edge-Disjoint} \((\leq k_1, *)\)-\textsc{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.** \textsc{Edge-Disjoint} \((\leq k_1, *)\)-\textsc{Paths} can be solved in \(2^{O(k_1 \log k_1)} m \log n\) time, and \textsc{Edge-Disjoint} \((= k_1, *)\)-\textsc{Paths} can be solved in \(2^{O(k_1 \log k_1)} m \log^2 n\) time.

**Proof.** We focus on \textsc{Edge-Disjoint} \((\leq k_1, *)\)-\textsc{Paths} as our analysis also works for \textsc{Edge-Disjoint} \((= k_1, *)\)-\textsc{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 \(\geq 1/2^{(k_1+1)^2}\). 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

\[
2^r O((\log r)^2) \log n \cdot m' = O(2.01^{k_1^2} m \log n).
\]
We can reduce $O(2.01^{t_1})$ 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 $2r(r - 1) + 4 \geq 2(r - 1)/2 + 2$. 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

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

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

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

3.2. One short and one long

We now consider EDGE-DISJOINT $(L_1, L_2)$-PATHS when $(L_1, L_2)$ is $(< k_1, > k_2)$ or $(= k_1, > k_2)$. 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 $P_2 = P_2[s_2, s^*]$ and $P'_2 = P_2[s^*, t_2]$. Obviously $P'_2$ can have at most $k_2 + 1$ nearby-vertices as it has $k_2 + 1$ vertices only. For nearby-vertices in $P'_2$, 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 $P'_2$ 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-$P'_2$ vertex) if it has a docking vertex in $P_1$ (resp., $P'_2$). Therefore every nearby-vertex is either near-$P_1$, near-$P'_2$, 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 $Q[v, v^*]\{v^*\}$ is always vertex-disjoint from both $P_1$ and $P'_2$.

We are ready to put a bound on the number of near-$P_1$ vertices in $P'_2$. For this purpose, we define for each vertex $x \in V(P_1) \cup V(P'_2)$ the following set of near-$P_1$ vertices:

$$D(x) = \{v : v \text{ is a near-$P_1$ vertex in } P'_2 \text{ and } x \text{ is a docking vertex of } v\}$$

Following the same arguments for $N^*_x$ in the proof of Lemma 1, we can use docking paths $Q[x, x]$ and $Q^{-1}[x, x]$, respectively, as paths $P_1$ and $P_2$ in that proof to show that $|D(x)| \leq k_1 + 1$. Therefore $P'_2$ contains at most $(k_1 + 1)^2$ near-$P_1$ vertices as $|V(P_1) \cup V(P'_2)| \leq k_1 + 1$.

Next we consider the number of near-$P'_2$ vertices in $P'_2$. Suppose that $P'_2$ contains at least $k_1 + k_2 + 2$ near-$P'_2$ vertices. Let $y$ be the $(k_1 + 2) \cdot (k_2 + 2)$-th near-$P'_2$ vertex in $P'_2$. Then there is a docking path $Q$ from some docking vertex $y'$ of $y$ in $P'_2$ to vertex $y$. Let $z$ be the last vertex of $P'_2$ that also appears in $Q$, and note that $z$ lies in $P_2[y, t_2]$. Denote the $(k_2 + 1)$-th
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 $P_2^s \setminus y'$, 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 $|P[s^*,y]| \geq |k_1|/2 + 1$ by the definition of $y$, we see that $|P[y',z]| > |Q[y',z]|$ 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 $P_2$. 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 $P_2^s$ can contain at most $k_1 + k_2 + 1$ near-$P_2$ vertices. Together with at most $(k_1 + 1)^2$ near-$P_1$ vertices in $P_2^s$ and $k_2 + 1$ vertices in $P_2$, 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. □

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 $2^{O(k_1 \log (k_1+k_2)+k_2)} m \log^2 n$ 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

$$O(t^6 \log m') \cdot \left(\frac{d}{k_1}\right) \cdot 8^{k_2+o(k_2)} m \log^2 n = 2^{O(k_1 \log (k_1+k_2)+k_2)} m \log^2 n.$$ 

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 $O(2.619^k m \log^2 n)$ time [14]. Therefore we obtain a deterministic FPT algorithm for the problem with running time

$$O(t^6 \log m') \cdot \left(\frac{d}{k_1}\right) \cdot O(2.619^k m \log^2 n + 8^{k_2+o(k_2)} m \log^2 n)$$

$$= 2^{O(k_1 \log (k_1+k_2)+k_2)} m \log^3 n.$$ □
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 \(\Pi\) to another parameterized problem \(\Pi'\) 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 \(\Pi\) if and only if \((I', k')\) is a yes-instance of \(\Pi'\), 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 \(\Pi\) to another parameterized problem \(\Pi'\), then \(\Pi'\) admits no polynomial compression (hence no polynomial kernel) whenever \(\Pi\) 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 \(\Pi\) takes \(p\) instances \((I_1, k), \ldots, (I_p, k) \in \Pi\) 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 \(\Pi\) if and only if some \((I_i, k)\) is a yes-instance of \(\Pi\), and

(b) \(k'\) is polynomial in \(\max_{i=1}^{p} |I_i| + \log p\).

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 \((\leq k_1, = k_2)\), \((\leq k_1, \geq k_2)\), \((= k_1, = k_2)\), \((= k_1, \geq k_2)\), \((\geq k_1, = k_2)\),\((\geq k_1, \geq k_2)\), or \((\geq k_1, \ast)\).

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., \(\geq k\)) if and only if one of \(G_i\) contains an \((s_i, t_i)\)-path of length \(k\) (resp., \(\geq k\)). By Theorem 6, this relaxed-composition establishes the lemma. \(\Box\)
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 $x_ss'$ and $x_ts''$, 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 $x_tt'$ 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 $RC(I)$ to compute a relaxed-composition of $I$. We may assume that $|I| = 2^d$ for some integer $d$, as we can always add some dummy no-instances to $I$.

Algorithm $RC(I)$.

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

If $I$ contains two instances $I'$ and $I''$ only

then return $I' \oplus I''$

else evenly split $I$ into $|I', I''|$ and return $RC(I') \oplus RC(I'')$.

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

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

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

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 $k_1, k_2 \leq n$. Also the construction of $I' \oplus I''$ takes time linear in $|I'| + |I''|$, and hence the algorithm constructs a relaxed-composition of $I$ in time linear in the total length of instances in $I$. Since the problem $\text{Edge-Disjoint (≤ k_1, ≤ k_2)-Paths}$ is NP-complete [28], it follows from Theorem 6 that the problem admits no polynomial compression (hence no polynomial kernel) unless $NP \subseteq \text{coNP/poly}$. □

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

5. Concluding remarks

We have obtained FPT algorithms to solve $\text{Edge-Disjoint (L_1, L_2)-Paths}$ 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 $\text{Edge-Disjoint (≥ k_1, *)-Paths}$ and $\text{Edge-Disjoint (≥ k_1, ≥ k_2)-Paths}$.

Since $\text{Edge-Disjoint (≥ k_1, *)-Paths}$ is equivalent to $\text{Edge-Disjoint (≥ k_1, ≥ k_2)-Paths}$ 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 $\text{Edge-Disjoint (≤ k_1, ≤ k_2)-Paths}$ 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 $(k_1, k_2)$, $(= k_1, k_2)$ 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 $\text{Edge-Disjoint (L_1, L_2)-Paths}$.

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 $\text{Vertex-Disjoint (L_1, L_2)-Paths}$ for undirected/directed graphs.

**Declaration of Competing Interest**

None declared.
Acknowledgements

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

References