Algorithmica (2020) 82:41–63
https://doi.org/10.1007/s00453-019-00602-8

A Polynomial Sized Kernel for Tracking Paths Problem

Aritra Banik[1] · Pratibha Choudhary[2] · Daniel Lokshtanov[3] · Venkatesh Raman[4,5] · Saket Saurabh[4,5]

Received: 27 March 2018 / Accepted: 21 June 2019 / Published online: 5 July 2019

Abstract
Consider a secure environment (say an airport) that has a unique entry and a unique exit point with multiple inter-crossing paths between them. We want to place (minimum number of) trackers (or check points) at some specific intersections so that based on the sequence of trackers a person has encountered, we can identify the exact path traversed by the person. Motivated by such applications, we study the TRACKING PATHS problem in this paper. Given an undirected graph with a source s, a destination t and a non-negative integer k, the goal is to find a set of at most k vertices, a tracking set, that intersects each s–t path in a unique sequence. Such a set enables a central controller to *track* all the paths from s to t. We first show that the problem is NP-complete. Then we show that finding a tracking set of size at most k is fixed-parameter tractable when parameterized by the solution size. More specifically, given an undirected graph on n vertices and an integer k, we give a polynomial time algorithm that either determines that the graph cannot be tracked by k trackers or produces an equivalent instance with $\mathcal{O}(k^7)$ edges.

Keywords Graph · s–t paths · Tracking paths · Parameterized complexity · FPT · Kernel · Feedback vertex set

1 Introduction

Tracking moving objects in a secure environment is an active area of research. Typically a secure environment is modelled as a network with fixed entry and exit point(s). Monitoring is achieved by placing sensor nodes which monitor the movements of the objects in the network. For a detailed study of field surveillance for the purpose of habitat monitoring, securing buildings, and intruder tracking please refer to [3,6]. While tracking traces of illegal activities over the Internet, the biggest challenge is to

✉ Pratibha Choudhary
pratibhac247@gmail.com

Extended author information available on the last page of the article

track moving data packets [13,15]. One may want to place trackers at an appropriate subset of routers in the network in order to track such activities.

Motivated by these applications, in a recent paper, Banik et al. [2] considered the problem of target tracking theoretically and modeled it as the following graph theoretic problem. Let $G = (V, E)$ be an undirected graph without any self loops or parallel edges, and suppose G has an unique entry vertex s and an unique exit vertex t. A simple path from s to t is called an s–t path. Let P be an s–t path in G and $T \subseteq V$ be a set of vertices. By λ_P^T we denote the sequence of vertices of T obtained from P by deleting the vertices that do not belong to T. T is a tracking set for G, if and only if for any two distinct s–t paths P_1 and P_2, $\lambda_{P_1}^T \neq \lambda_{P_2}^T$. The vertices in set T are called trackers. Banik et al. [2] proved that the problem of finding a minimum-cardinality tracking set with respect to the *shortest* s–t paths (TRACKING SHORTEST PATHS) is NP-hard and APX-hard. In this paper we consider the problem of tracking all simple paths and not just the shortest paths. In particular, we study the following problem.

TRACKING PATHS (G, s, t, k) **Parameter:** k
Input: An undirected graph $G = (V, E)$ with two distinguished vertices s and t, and a non-negative integer k.
Question: Is there a tracking set T of size at most k for G?

Observe that if we consider all s–t paths and not just the simple s–t paths, then the solution is quite simple. We first retain only those vertices and edges that are reachable from s and t, and delete the remaining vertices and edges. Now the set of all remaining vertices form a tracking set. Note that any path that traverses an edge back and forth, more than once, makes it necessary for both end points of the edge to be marked as trackers. Hence, the set of all vertices reachable from both s and t, forms an optimum tracking set, when we consider all s–t paths, including the paths that are not simple.

Our Results and Methods. In this paper we study TRACKING PATHS from the perspective of parameterized complexity. Our first contribution is the following theorem.

Theorem 1 TRACKING PATHS *is NP-complete.*

As it is the case for any proof of NP-completeness, the proof of Theorem 1 requires two steps: hardness, and, containment inside NP. While hardness follows by a slight modification to a result of Banik et al. [2] (see Lemma 1), it is not immediately clear why the problem is in NP. To check whether a given set of vertices is a tracking set for G, we need to go over all pairs of paths between s and t, and check whether the sequence of trackers in each of them is unique. However, the number of paths between s and t could be exponentially large, and thus it does not seem possible to exploit the definition of the problem in order to show its containment in NP. Thus, to show that the problem belongs to NP, we first give an alternate characterization for a tracking set. Further, we show that T is a *feedback vertex set* (FVS) for G. That is, $G \setminus T$ is a forest. Using this property and our alternate characterization of tracking set we give a polynomial time algorithm to test whether a given set of vertices is a tracking set.

Once we have shown that TRACKING PATHS is NP-complete, we study the problem from the viewpoint of parameterized complexity. In particular we design a fixed-parameter tractable (FPT) algorithm for TRACKING PATHS. That is, we design an algorithm for TRACKING PATHS with running time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$, where k is the size of the tracking set we seek. In fact, we give a stronger result than just designing an FPT algorithm; we give a polynomial kernel for the problem. In particular, given an instance (G, s, t, k), we give a polynomial time algorithm that either determines that (G, s, t, k) is a NO instance or produces an equivalent instance with $\mathcal{O}(k^6)$ vertices and $\mathcal{O}(k^7)$ edges. This polynomial time algorithm is called a *kernelization algorithm* and the reduced instance is called a *kernel*. For more details about parameterized complexity and kernelization we refer to monographs [4,5]. Our second contribution is the following result.

Theorem 2 TRACKING PATHS *admits a polynomial kernel of size* $\mathcal{O}(k^7)$.

The kernelization algorithm (proof of Theorem 2) works along the following lines. Let (G, s, t, k) be an input instance to TRACKING PATHS. We first apply a simple reduction rule to ensure that each edge and vertex in G belongs to some s–t path. Then we prove two structural claims: (a) every tracking set S is an FVS; and (b) if there exists a pair of vertices x and y in G, such that x and y have at least $k + 4$ vertex disjoint paths between them, then (G, s, t, k) is a NO instance.

Next, using the known factor 2-approximation algorithm for the FEEDBACK VERTEX SET problem, we compute a set S such that $G \setminus S$ is a forest. If $|S| > 2k$ then we immediately return that (G, s, t, k) is a NO instance. Else, we assume that the size of S is at most $2k$. Now using the second structural claim regarding tracking set, we show that the number of connected components in $G \setminus S$, and the number of vertices in $V(G \setminus S)$ that have at least two neighbors in S, is at most $k^{\mathcal{O}(1)}$. Next, we bound the number of vertices in $V(G \setminus S)$ that have *exactly* one neighbor in S. To do this, we fix a tree R in $G \setminus S$ and a vertex $v \in S$, and bound the size of the set of neighbors of v, $N_R(v)$, in R. Towards this, we consider the minimal subtree T in R that contains all the neighbors of v, and show that if $|N_R(v)| > k^{\mathcal{O}(1)}$, then we can partition the tree T into $k + 1$ parts in such a way that each part must contain a vertex marked as a tracker. This bounds the degree of each vertex in S into $V(G \setminus S)$ by $k^{\mathcal{O}(1)}$. This, together with some well-known counting methods on trees, gives us the desired polynomial kernel for TRACKING PATHS.

Related Work. Different structural properties of graphs have been studied previously to analyze navigational models in network settings. In a seminal paper, Slater [14] introduced the concept of metric dimension of a graph. In graph theory, the metric dimension of a graph G is the minimum cardinality of a subset S of vertices such that all other vertices are uniquely determined by their distances to the vertices in S [7]. One application of metric dimension is the problem of determining the location of an object in a network, depending on its distance from different landmarks in the network [11]. For a survey of metric dimension in graphs see [8]. Furthermore, as mentioned before TRACKING PATHS is also related to FEEDBACK VERTEX SET in a graph. FEEDBACK VERTEX SET is NP-hard [9], has a 2 approximation algorithm [1], a quadratic sized kernel [16] and an FPT algorithm running in time $\mathcal{O}((3.619)^k n^{\mathcal{O}(1)})$ [12].

1.1 Preliminaries

A kernelization algorithm is typically obtained using what are called *reduction rules*. These rules transform a given parameterized instance to another equivalent instance in polynomial time. A rule is said to be *safe* if the resulting graph has a tracking set of size at most k if and only if the original instance has one.

As mentioned earlier, when considering tracking set for a graph $G = (V, E)$, we assume that we are given a unique source $s \in V$ and a unique destination $t \in V$, and we aim to find a tracking set that can distinguish between all simple paths between s and t. Here s and t are also referred to as the terminal vertices. We say that a vertex v is a tracker, if v has been marked as a tracker. If $a, b \in V$, then unless otherwise stated, $\{a, b\}$ represents the set of vertices a and b, and (a, b) represents an edge between a and b. For a vertex, $v \in V$, its *neighborhood*, $N(v) \subseteq V$ consists of the set of vertices that are adjacent to v.

For a path P, $V(P)$ denotes the vertex set of path P. For a subgraph (or graph) G', $V(G')$ denotes the vertex set of G', and $E(G')$ denotes the edge set of G'. Let P_1 be a path between vertices a and b, and P_2 be a path between vertices b and c, such that $V(P_1) \cap V(P_2) = \{b\}$. Then, by $P_1 \cdot P_2$, we denote the path between a and c, formed by concatenating paths P_1 and P_2 at vertex b. Two paths P_1 and P_2 are said to be *vertex disjoint* if their vertex sets do not intersect, except possibly at the end points, i.e. $V(P_1) \cap V(P_2) \subseteq \{a, b\}$, where a and b appear only as ending points of one or both of the paths. If $G = (V, E)$ is a graph, then $G \backslash V'$ depicts the graph induced by the vertex set $V \backslash V'$, and $(G \backslash V') \cup V''$ depicts the graph induced by the vertex set $(V \backslash V') \cup V''$, where $V', V'' \subseteq V$. P_{ab} denotes a simple path between vertices a and b, where $a, b \in V$. For a path P, if $x, y \in V(P)$, then by $P[x, y]$, we denote the segment of path P lying between the vertices x and y. For two paths P_1 and P_2, we say $P_1 = P_2$, if both P_1 and P_2 consist of the same sequence of vertices. $P_1 \cup P_2$ denotes the graph induced by the vertices in $V(P_1) \cup V(P_2)$. Similarly, $P_1[a, b] \cup P_2[c, d]$ denotes the graph induced by vertices in segments of paths P_1 and P_2, lying between vertices a, b and c, d respectively.

2 NP-Completeness

In this section we show that TRACKING PATHS is NP-complete. We first show that the problem is NP-hard. Banik et al. [2] showed TRACKING SHORTEST PATHS to be NP-hard by giving a reduction from VERTEX COVER to TRACKING SHORTEST PATHS. We show that with some minor modification, the same reduction also proves TRACKING PATHS to be NP-hard.

Lemma 1 *Let (G, k) be an instance of* VERTEX COVER. *Then there exists an instance (G', k') of* TRACKING PATHS, *such that G has a vertex cover of size k if and only if G' has a tracking set (for all paths) of size $k' = k + |E| + 2$.*

Proof Let $\{G = (V, E), k\}$ be an instance of VERTEX COVER. We construct graph $G' = (V', E')$ from $G = (V, E)$ as follows. Add a vertex in G' for each vertex in G, and for each edge in G. Also add vertices s and t in G'. Now $V' = V_r \cup E_r \cup \{s, t\}$,

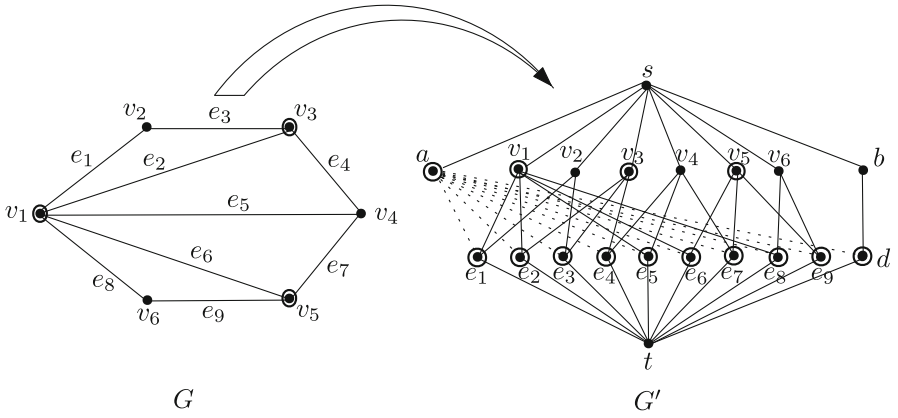


Fig. 1 Reduction from VERTEX COVER to TRACKING PATHS (circled vertices represent vertex cover in G, and tracking set in G')

where $V_r = \{u_v \mid v \in V\}$ and $E_r = \{u_e \mid e \in E\}$. In G', we introduce an edge between a vertex $u_v \in V_r$ and a vertex $u_e \in E_r$ if the edge corresponding to e in G, is incident to v in G. Also introduce an edge between u_v and s, for each $u_v \in V_r$, and introduce an edge between u_e and t, for each $u_e \in E_r$, as shown in Fig. 1. Add the vertices a, b, and d to V', and the edges (s, a), (s, b), and (d, t) to E'. Also add the edges (a, x), for each $x \in E$, and the edges (a, d) and (b, d) to E' (see Fig. 1).

First we prove that if V_c is a vertex cover for G, then $T = V_c \cup E_r \cup \{a, d\}$ is a tracking set for all s–t paths in G'. Suppose not. Then there exist two distinct s–t paths (not necessarily shortest), say P_1 and P_2 in G', such that they contain the same sequence of trackers. Since P_1 and P_2 are two distinct paths, there exists a vertex that is present in exactly one of these two paths, but not in the other. Without loss of generality, assume that there exists a vertex v such that, $v \in V(P_1) \backslash V(P_2)$. Observe that, $v \in V_r \backslash V_c \cup \{b\}$. First we consider the case when, $v = b$. Note that due to the structure of G', b is followed by d in every path that contains b. Hence, P_1 contains d. Since d is a tracker, P_2 must contain a sequence of vertices that includes the vertex d. The only path possible that would not contain b, but would contain d, necessarily includes the vertex a followed by the vertex d. Notice that since a is a tracker itself, this contradicts the assumption that P_1 and P_2 contain the same sequence of trackers. Next we consider the case when $v \in V_r \backslash V_c$. Observe that any path consisting of such a vertex v, would contain v followed by some vertex, say $e' \in E_r$. Since e' is a tracker, and P_1 and P_2 contain the same sequence of trackers, $e' \in V(P_1)$, and $e' \in V(P_2)$. Hence, in order for P_2 to be distinct from P_1, there must exist another vertex $v' \in V_r \backslash V_c$, such that P_2 contains the vertex v' followed by e'. Thus the edge corresponding to e' in G, is incident to the vertices corresponding to v and v' in G. Hence at least one among v and v' necessarily belongs to V_c, and, consequently is a tracker. This contradicts the assumption that P_1 and P_2 contain the same sequence of trackers.

Next we prove that if there exists a tracking set (for all s–t paths) of size $k + |E| + 2$ in G', then there exists a vertex cover of size k in G. First, note that for each vertex

$e_i \in E_r$, there exists two vertex disjoint paths between s and e_i, that pass through the vertices in V_r that correspond to the vertices that were the end points of the edge corresponding to e_i in G. And, in order to distinguish between these paths, at least one of the vertices that corresponded to the end points of the edge e_i, necessarily belong to T. Since, this holds for each $e_i \in E_r$, we have that, if there exists a vertex cover of size k in G, then k vertices (V_c) among V_r belong to T. Now consider the graph G'' induced on $V(G')\setminus V_c$. Note that there exists two vertex disjoint paths between s and each $e_j \in E_r$, one passing through a and another passing through a vertex in $V_r \setminus V_c$. Hence either a or all of $V_r \cap V(G'')$ belongs to T. Without loss of generality, assume that a belongs to T. Next, observe that there exist two paths from s to a, one via the edge (s, a), and another via vertices b and then d. Hence, at least one among b and d have to be part of tracking set. Note that since a is incident to all vertices in $E_r \cup \{d\}$, and all these vertices are adjacent to t, there are $|E_r| + 1$ vertex disjoint paths between a and t. Hence, by pigeonhole principle, all but one vertices in $E_r \cup \{d\}$ necessarily need to be trackers. Hence, a tracking set T of size $k + |E| + 2$ in G' necessarily consists of $V_c \cup E_r \cup \{d\}$. Note that, d can be replaced by b here. By the arguments given for the vertices contained in T, it can be seen that if there exists a tracking set of size $k + |E| + 2$ in G', then there exists a vertex cover of size k in G. Hence, the lemma holds. □

Theorem 3 TRACKING PATHS *is NP-hard.*

Proof Lemma 1 proves that there exists a polynomial time reduction from VERTEX COVER to TRACKING PATHS. NP- hardness of TRACKING PATHS follows since VERTEX COVER is NP- hard [9]. □

In what follows, we show that TRACKING PATHS is in NP. Towards that we first give an alternative characterization for a tracking set. Then using a preprocessing rule, we show that for a graph, every tracking set is also a feedback vertex set (FVS). Finally using these two properties we devise a polynomial time algorithm to check whether a given set of vertices is a tracking set for graph G or not.

2.1 Characterization of Tracking Set

Towards characterization of a tracking set, we first define the *tracking set condition*.

Tracking Set Condition:
For a graph $G = (V, E)$, with terminal vertices $s, t \in V$, a set of vertices $T \subseteq V$, is said to satisfy the *tracking set condition* if there does not exist a pair of vertices $u, v \in V$, such that the following holds:

- there exist two distinct paths, say P^1_{uv} and P^2_{uv}, between u and v in $(G\setminus(T \cup \{s, t\})) \cup \{u, v\}$, and
- there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt}, in $(G\setminus(V(P^1_{uv}) \cup V(P^2_{uv}))) \cup \{u, v\}$, and $V(P_{su}) \cap V(P_{vt}) = \emptyset$, i.e. P_{su} and P_{vt} are mutually vertex disjoint, and also vertex disjoint from P^1_{uv} and P^2_{uv}.

See Figs. 2 and 3.

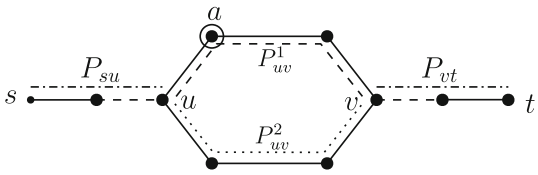


Fig. 2 *Tracking Set Condition* satisfied (circled vertex represents a tracker)

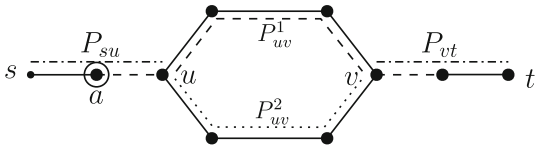


Fig. 3 *Tracking Set Condition* not satisfied (circled vertex represents a tracker)

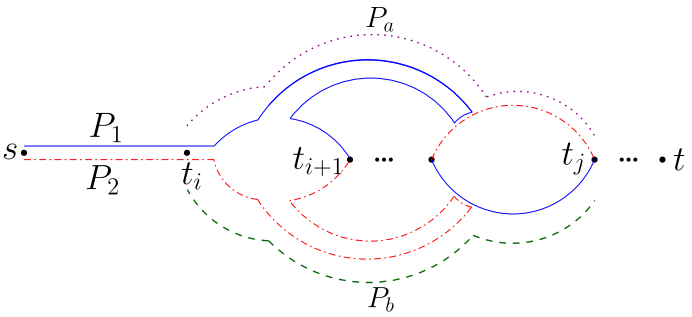


Fig. 4 Paths P_1 and P_2 are marked using straight line and dash dotted line, respectively. Paths P_a and P_b are shown using dotted line and dashed line, respectively

Next we have a lemma that establishes the relation between the *tracking set condition* and a tracking set.

Lemma 2 *For a graph $G = (V, E)$ with terminal vertices $s, t \in V$, a set of vertices $T \subseteq V$ is a tracking set if and only if T satisfies the tracking set condition.*

Proof Let us assume that $T \subseteq V$ is a tracking set for G. We claim that T satisfies the *tracking set condition*. Suppose not. Then there exists a pair of vertices $u, v \in V$, such that there exist two distinct paths, say P^1_{uv}, P^2_{uv}, between u and v in $(G \backslash (T \cup \{s, t\})) \cup \{u, v\}$, and, there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt} in $(G \backslash (V(P^1_{uv}) \cup V(P^2_{uv}))) \cup \{u, v\}$, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$. Notice that there might be trackers in $V(P_{su}) \cup V(P_{vt})$, however there are no trackers in $(V(P^1_{uv}) \cup V(P^2_{uv})) \backslash \{u, v\}$. Observe now there exist two s–t paths in G, $P_1 = P_{su} \cdot P^1_{uv} \cdot P_{vt}$, and, $P_2 = P_{su} \cdot P^2_{uv} \cdot P_{vt}$, such that both have the same sequence of trackers. This contradicts the assumption that T is a tracking set for G.

Conversely, let us assume that there exists a set of vertices, $T \subseteq V$, such that T satisfies the *tracking set condition*. We claim that T is a tracking set for G. Suppose not. Then there exist at least two distinct s–t paths in G, say P_1 and P_2, that contain the same sequence of trackers.

Let the sequence of trackers that appears in P_1 and P_2 be $T = (t_1, \ldots, t_k)$. We define $T' = T \cup \{s\} \cup \{t\}$. Let us assume that t_i be the first vertex in T', such that

$P_1[s, t_i] = P_2[s, t_i]$ and $P_1[t_i, t_{i+1}] \neq P_2[t_i, t_{i+1}]$. See Fig. 4, colors have been used for better visibility of paths. Observe that, it may be the case that, $t_i = s$.

Let $j > i$ be the maximum index such that

- there exist two paths, P_a and P_b, between t_i and t_j, such that $P_a \neq P_b$, where $t_j \in T'$ (subscripts a and b have been used just to identify the paths)
- P_a and P_b do not contain any trackers other than t_i and t_j.
- $P_a \subseteq P_1[t_i, t_j] \cup P_2[t_i, t_j]$ and $P_b \subseteq P_1[t_i, t_j] \cup P_2[t_i, t_j]$.

Observe that such an index j always exists as $j = i + 1$ satisfies all the above conditions (see Fig. 4).

Claim 1 There exists a path between t_j and t in $(P_1[t_j, t] \cup P_2[t_j, t])\backslash(P_a \cup P_b)$.

Proof For the sake of contradiction, we assume that there does not exist a path between t_j and t in $(P_1[t_j, t] \cup P_2[t_j, t])\backslash(P_a \cup P_b)$.

Observe that for any $l > j$, there exist two paths (not necessarily disjoint), $P_1[t_l, t_{l+1}]$ and $P_2[t_l, t_{l+1}]$, between t_l and t_{l+1}. There does not exist a path between t_l and t_{l+1}, in $(P_1[t_l, t_{l+1}] \cup P_2[t_l, t_{l+1}])\backslash(P_a \cup P_b)$, if and only if, $P_a \cup P_b$ intersects both $P_1[t_l, t_{l+1}]$ and $P_2[t_l, t_{l+1}]$. Therefore, if there does not exist a path between t_j and t in $(P_1[t_j, t] \cup P_2[t_j, t])\backslash(P_a \cup P_b)$, then there must exist two consecutive trackers t_r and t_{r+1}, where $r \geq j$, such that $P_1[t_r, t_{r+1}]$ and $P_2[t_r, t_{r+1}]$ have a nonempty intersection with $(P_a \cup P_b)$.

Let $\lambda_1 = P_1[t_r, t_{r+1}] \cap (P_a \cup P_b)$, and $\lambda_2 = P_2[t_r, t_{r+1}] \cap (P_a \cup P_b)$ (see Fig. 5, colors have been used for better visibility of paths). Observe that $P_1[t_r, t_{r+1}]$ and $P_2[t_r, t_{r+1}]$ are not necessarily disjoint. However, we claim that λ_1 and λ_2 are disjoint. If not, then suppose $p \in \lambda_1 \cap \lambda_2$. Hence, $p \in P_1[t_r, t_{r+1}] \cap P_2[t_r, t_{r+1}]$. But $p \in P_a \cup P_b$. Hence, $p \in P_1[t_i, t_j] \cup P_2[t_i, t_j]$. Without loss of generality, assume that $p \in P_1[t_i, t_j]$. Thus, $p \in P_1[t_r, t_{r+1}]$, and $p \in P_1[t_i, t_j]$. But, this contradicts the fact that P_1 is a simple path. For the rest of the proof, we assume that λ_1 and λ_2 are vertex disjoint paths. Observe that following are the two possible cases.

Case (i): Both λ_1 and λ_2 intersect P_a, or both λ_1 and λ_2 intersect P_b.
Case (ii): λ_1 intersects P_a, but does not intersect P_b, and, λ_2 intersects P_b, but does not intersect P_a.

We first consider Case (i). See Fig. 5. Let c and d be the first points appearing in $\lambda_1 \cap P_a$ and $\lambda_2 \cap P_a$, respectively. Without loss of generality, assume that c appears before d in P_a. Let $P_a^* = P_a[t_i, c] \cdot P_1[c, t_{r+1}]$, and $P_b^* = P_a[t_i, d] \cdot P_2[d, t_{r+1}]$. Observe that, $P_a^* \neq P_b^*$ as $c \notin P_b^*$, and $d \notin P_a^*$. This contradicts the maximality of j, since now $r + 1 > j$, and there exist two paths, P_a^* and P_b^*, between t_i and t_{r+1}, and $P_a^* \neq P_b^*$, and these two paths do not contain any trackers other than t_i and t_{r+1}. Hence the claim holds.

Next we consider Case (ii). Let c and d be the first points appearing in $\lambda_1 \cap P_a$ and $\lambda_2 \cap P_b$, respectively. Let $P_a^* = P_a[t_i, c] \cdot P_1[c, t_{r+1}]$, and $P_b^* = P_b[t_i, d] \cdot P_2[d, t_{r+1}]$. Observe $P_a^* \neq P_b^*$ as $c \notin P_b^*$, and $d \notin P_a^*$. This contradicts the maximality of j, since now $r + 1 > j$, and there exist two paths, P_a^* and P_b^*, between t_i and t_{r+1}, and $P_a^* \neq P_b^*$, and these two paths do not contain any trackers other than t_i and t_{r+1}. Hence the claim holds. □

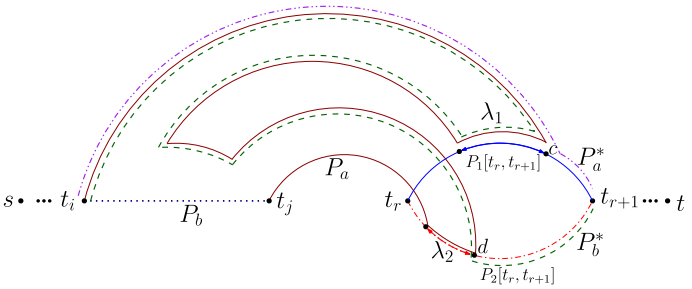


Fig. 5 Paths P_a and P_b are indicated using straight line and dotted line, respectively. Paths P_a^* and P_b^* are indicated using dash dot dotted line and dashed line, respectively

Next we show the *tracking set condition* being violated by proving the existence of two vertices $u, v \in V$, such that the following holds:

- there exist two distinct paths, say P_{uv}^1 and P_{uv}^2, between u and v, in $(G \backslash (T \cup \{s, t\})) \cup \{u, v\}$, and
- there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt}, in $(G \backslash (V(P_{uv}^1) \cup V(P_{uv}^2))) \cup \{u, v\}$, and $V(P_{su}) \cap V(P_{vt}) = \emptyset$.

Consider the graph G_{ab} induced by $V(P_a) \cup V(P_b)$. Let $P[s, t_i] = P_1[s, t_i]$. Also, there exists a path between t_j and t in $(P_1[t_j, t] \cup P_2[t_j, t]) \backslash (P_a \cup P_b)$. We denote this path by $P[t_j, t]$. Observe that P_a and P_b are two paths between t_i and t_j, where $P_a \neq P_b$, and $G_{ab} \backslash \{t_i, t_j\}$ does not contain any trackers. Therefore there must exist a cycle C in G_{ab}. Observe that there exists at least one edge in P_1 that is also present in C, otherwise C will be fully contained in P_2 which contradicts the fact that P_2 is a simple path. Let u and v be the first and last vertices from C visited by P_1. Observe that there are two paths P_{uv}^1 and P_{uv}^2, between u and v in C. Let $P_{su} = P[s, t_i] \cup P_1[t_i, u]$, and $P_{vt} = P_1[v, t_j] \cup P[t_j, t]$. Observe that the existence of P_{uv}^1, P_{uv}^2, P_{su} and P_{vt}, together contradict the *tracking set condition* and hence the result holds. □

Although we have a nice characterization for what qualifies to be a tracking set, we cannot use it yet for polynomial time verification, since there can be exponentially many paths between s and t in an arbitrary graph. However, in the next subsection, we show that in our case we can assume that between any two vertices, we have only polynomially many relevant paths, if we remove all the trackers along with s and t from the graph.

2.2 Tracking Set as Feedback Vertex Set

Let (G, s, t, k) be any instance of TRACKING PATHS. After applying a reduction rule that ensures that each edge and vertex in G belongs to some s–t path, we show that every tracking set is also an FVS for the reduced graph. For a graph $G = (V, E)$, an FVS is a set of vertices $S \subseteq V$, such that $G \backslash S$ is a forest.

Reduction Rule 1 *If there exists a vertex or an edge in G that does not participate in any s–t path, then delete it.*

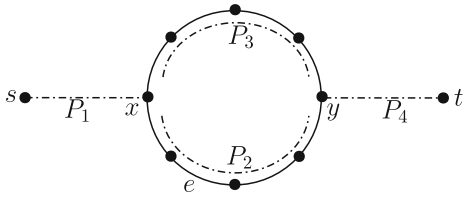


Fig. 6 Cycle without tracker leads to two *s*-*t* paths with same sequence of trackers

Lemma 3 *Reduction Rule 1 is safe and can be implemented in polynomial time.*

Proof Vertices and edges that do not participate in any *s*–*t* path, cannot help distinguish between two *s*–*t* paths in a graph. Thus, after removal of such vertices and edges, a tracking set for the resulting graph will also be a tracking set for the original graph and vice versa. Hence, Reduction Rule 1 is safe.

An edge $e = (u, v)$ participates in an *s*–*t* path if and only if there exist two simple paths, one from s to u and another from v to t, such that both these paths are mutually vertex disjoint (or simple paths from s to v and from u to t that are mutually vertex disjoint). Existence of such paths can be determined in quadratic time [10]. Note that removing all edges that do not participate in any *s*–*t* path will isolate the vertices that do not participate in any *s*–*t* path. In linear time we can remove such vertices from the graph. □

In the remainder of this paper we assume that each vertex and each edge participates in at least one *s*–*t* path. Next we have a lemma that establishes the connection between a tracking set and an FVS.

Lemma 4 *If T is a tracking set for G then T is a feedback vertex set for G.*

Proof Consider a cycle C in G. We show that T contains at least one vertex from C. Consider an edge e in cycle C. Since every edge in the graph participates in at least one *s*–*t* path, there exists an *s*–*t* path, say P, that contains the edge e. Path P may contain some more vertices and edges from the cycle C. Let x be the first vertex of C that appears in path P while traversing from s to t. Similarly let y be the last vertex of C that appears in path P while traversing from s to t (see Fig. 6). Observe that there are two paths between x and y in cycle C, one of them containing edge e, and the other one not containing edge e. Denote the path containing the edge e by P_2, and the other path by P_3. Let P_1 be the subpath (which would contain only s if s is in the cycle C) of P from s to x, and P_4 be the subpath (which would contain only t if t is in the cycle C) of P from y to t. Consider the following two paths:

$$P' = P_1 \cdot P_2 \cdot P_4$$
$$P'' = P_1 \cdot P_3 \cdot P_4$$

Observe that if C does not contain any trackers, then P' and P'' contain exactly the same sequence of trackers, thus contradicting the fact that T is a tracking set. □

Thus we have the following corollary.

Corollary 1 *The size of a minimum tracking set for a graph G is at least the size of a minimum FVS for G.*

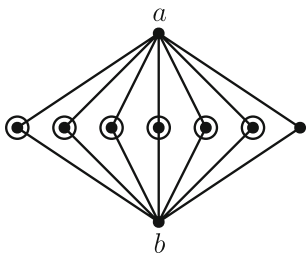


Fig. 7 Feedback Vertex Set versus Tracking Set (circled vertices represent trackers)

Note that although a tracking set is also an FVS for a graph, they are very different. Consider the example of a graph G induced by m vertex disjoint paths, with a common starting point a, and a common ending point b. Here a single vertex, a (or b) is an FVS for G. But we need exactly $m-1$ trackers in $G\backslash\{a,b\}$, one each on all but one vertex disjoint paths, in order to track all paths between a and b. See Fig. 7. Hence, the cardinality of a minimum tracking set can be arbitrarily larger than the cardinality of a minimum FVS for a graph.

2.3 Verification of Tracking Set

From Lemma 2, we know that a set $T \subseteq V$ of vertices is a tracking set for graph $G = (V, E)$, if and only if it satisfies the *tracking set condition*. Now for each pair of vertices $u, v \in V$, we give a bound on the number of paths between them in $(G\backslash(T \cup \{s,t\})) \cup \{u,v\}$.

Lemma 5 *Let T be a tracking set for graph $G = (V, E)$, $|V| = n$, and let u, v be a pair of distinct vertices in V, then the number of distinct paths between u and v in $(G\backslash(T \cup \{s,t\})) \cup \{u,v\}$ is $\mathcal{O}(n^2)$, and they can be found in $\mathcal{O}(n^3)$ time.*

Proof From Lemma 4, we know that if T is a tracking set for graph $G = (V, E)$, then T is also an FVS for G. Consider a pair of vertices $u, v \in V$. Let $V' = V\backslash(T \cup \{s,t\})$, and G' be the graph induced on V'. Note that G' is a forest. Let $V'' = V' \cup \{u,v\}$, and G'' be the graph induced on V''. Next we consider the following possible cases:

- $u, v \in V'$. Since G' is a forest, there exists at most one path between u and v in G''.
- $u, v \in V\backslash V'$, i.e., $u, v \in T \cup \{s,t\}$. Observe that u and v can have at most $n-1$ neighbors each, in G'', and $N_{G''}(u) \subseteq V' \cup \{v\}$, and $N_{G''}(v) \subseteq V' \cup \{u\}$. Since G' is a forest, there exists at most one path between each pair of neighbors in V'. Since the paths between u and v either pass through V', or consists of edge (u,v), the maximum number of paths between u and v is at most $(n-1)^2+1$. Specifically there can exist at most 1 path of length one, $n-2$ paths of length two, and $(n-1)^2$ paths of length at least three. Hence the number of paths between u and v is $\mathcal{O}(n^2)$.
- $u \in V'$ and $v \in V\backslash V'$. Since G' is a forest, and v can have at most $n-1$ neighbors in G'', there can be at most $n-1$ paths between u and v in G''. Note that the case when $v \in V'$ and $u \in V\backslash V'$ is symmetric to the current case.

The path between a pair of vertices in G' can be found using depth first search in $\mathcal{O}(n)$ time since G' is a forest. Thus finding all distinct paths between u and v, in $G' \cup \{u, v\}$, can be done in $\mathcal{O}(n^3)$ time. □

Given a graph G, $|V(G)| = n$, and, two pairs of vertices, say u, v and u_1, v_1, we can verify if there exist two vertex disjoint paths, from u to u_1, and from v to v_1, in $\mathcal{O}(n^2)$ time, using the algorithm for vertex disjoint paths by Kawarabayashi et al. [10]. Hence we have the following lemma.

Lemma 6 *In a graph G, $|V(G)| = n$, with terminal vertices $s, t \in V(G)$, for a pair of vertices $u, v \in V(G)$ verifying if there exists a path from s to u, P_{su} and a path from v to t, P_{vt}, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$, can be done in $\mathcal{O}(n^2)$ time.*

Algorithm 1: Verifying a Tracking Set T.

Input: Instance of the TRACKING PATHS problem $(G = (V, E), s, t, k), T \subseteq V$
Output: YES if T is a Tracking Set, else NO

```
1  if G\T has a cycle then
2  |  return NO;
3  end
4  G_1 = G\(T ∪ {s, t}) ;
5  foreach pair of vertices u, v ∈ V do
6  |  if u ∉ V(G_1) OR v ∉ V(G_1) then
   |  |  // If u, v ∈ V(G_1), there exists at most one path between u and
   |  |     v since G_1 is a forest
7  |  |  G_2 = (G\(T ∪ {s, t})) ∪ {u, v};
8  |  |  if ∃ two or more paths between u, v in G_2 then
9  |  |  |  P = set of paths between u, v in G_2 ;
10 |  |  |  foreach pair of paths P_1, P_2 ∈ P do
11 |  |  |  |  G_3 = (G\(V(P_1) ∪ V(P_2))) ∪ {u, v};
12 |  |  |  |  if ∃ a path from s to u, P_su, and a path from v to t, P_vt, in G_3, such that
   |  |  |  |  V(P_su) ∩ V(P_vt) = ∅ then
13 |  |  |  |  |  return NO;
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 return YES;
```

Lemma 7 *For a graph $G = (V, E)$, and a set of vertices $T \subseteq V$, it can be determined in polynomial time whether T is a tracking set for all s–t paths in G.*

Proof From Lemma 4, we know that if T is a tracking set for G, then T is also an FVS for G. Thus, Algorithm 1 first checks if T is an FVS for G. If not, then it returns NO. Henceforth, it is assumed that T is an FVS for G. From Lemma 2, we know that a set of vertices is a tracking set if and only if it satisfies the *tracking set condition*. We

define $G_1 = G \setminus (T \cup \{s, t\})$. As required by the *tracking set condition*, the algorithm correctly considers each pair of vertices, say $u, v \in V$, and checks if there exist two or more distinct paths, say P_1 and P_2 between u and v in $(G \setminus (T \cup \{s, t\})) \cup \{u, v\}$. If yes, then it checks if there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt}, in $(G \setminus (V(P_1) \cup V(P_2))) \cup \{u, v\}$, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$. If yes, then the algorithm returns NO. Else, if no such paths are found, then the algorithm returns YES.

Let $|V| = n$, and $|E| = m$. Using breadth first search, in $\mathcal{O}(m + n)$ time we can check whether $G \setminus T$ contains a cycle and verify if T is an FVS for G. For each pair of vertices $u, v \in V$, we consider the graph $G_2 = (G \setminus (T \cup \{s, t\})) \cup \{u, v\}$. Note that construction of graph G_2 takes $\mathcal{O}(n^2)$ time assuming adjacency matrix representation. From Lemma 5, we know that there can be $\mathcal{O}(n^2)$ paths between u and v in G_2, and these can be found in $\mathcal{O}(n^3)$ time. For each pair of paths, P_1 and P_2, among the $\mathcal{O}(n^2)$ paths in $\mathcal{P}$, we consider the graph $G_3 = (G \setminus (V(P_1) \cup V(P_2))) \cup \{u, v\}$. Note that there exist $\mathcal{O}(n^4)$ graphs to consider for G_3, and each of these graphs can be constructed in $\mathcal{O}(n^2)$ time (assuming adjacency matrix representation). In G_3, using Lemma 6, we can check if there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt}, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$, in $\mathcal{O}(n^2)$ time. Hence the overall time taken for verification is $\mathcal{O}(n^2(n^2 + n^3 + n^4(n^2 + n^2)))$. Thus Algorithm 1 runs in $\mathcal{O}(n^8)$ time. □

From Lemmas 6 and 7, we have the following corollary.

Corollary 2 TRACKING PATHS *is in NP.*

Lemma 1 and Corollary 2 together prove Theorem 1.

3 Polynomial Kernel for TRACKING PATHS

In this section, with the help of some reduction rules we give a polynomial time algorithm that checks whether the given instance is a NO instance (for a solution of size at most k), or produces an equivalent instance with $\mathcal{O}(k^6)$ vertices and $\mathcal{O}(k^7)$ edges. We assume that the given graph has been preprocessed using Reduction Rule 1.

Recall from Corollary 1 that the size of a minimum tracking set T for G is at least the size of a minimum FVS for G. We start with a 2-approximate solution for FEEDBACK VERTEX SET S in G using the algorithm of Bafna et al. [1]. From Corollary 1, we have the following reduction rule.

Reduction Rule 2 *Apply the algorithm of Bafna et al.* [1] *to find a 2-approximate solution S for* FEEDBACK VERTEX SET. *If* $|S| > 2k$, *then return that the given instance is a NO instance.*

Observe that $\mathcal{F} = G \setminus S$ is a forest. Now we try to bound the number of vertices in graph $\mathcal{F}$ when k trackers are sufficient to track all s–t paths in G. Towards this we first prove a monotonicity lemma and a corollary which says that if a subgraph of G cannot be tracked with k trackers, then G cannot be tracked with k trackers either.

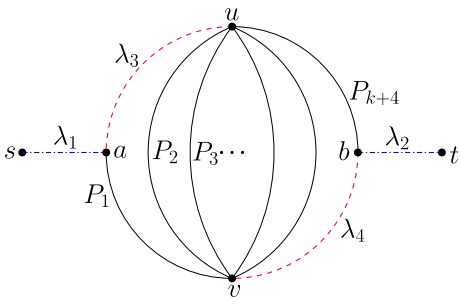


Fig. 8 Illustration of Lemma 9: $k+4$ vertex disjoint paths between pair of vertices u and v

Lemma 8 *Let $G = (V, E)$ be a graph, and $G' = (V', E')$ be a subgraph of G such that $s, t \in V'$. If T is a tracking set for G, then T is also a tracking set for G'.*

Proof We show that T is a tracking set for G' as well. Suppose not. Then there must exist two s–t paths, say P_1 and P_2, in G' that contain the same sequence of trackers. Observe that P_1 and P_2 also belong to G. Hence, P_1 and P_2 cannot be distinguished by T in G as well. This contradicts the assumption that T is a tracking set for G. Hence the lemma holds. □

Corollary 3 *If a subgraph of G that contains both s and t cannot be tracked by k trackers, then G cannot be tracked by k trackers either.*

Henceforth, we limit ourselves to analyzing the cases when a subgraph of G cannot be tracked by at most k trackers. We first prove a lemma that bounds the number of vertex disjoint paths between a pair of vertices in G if it is a YES instance.

Lemma 9 *If there are two vertices $u, v \in V$ such that there exist at least $k+4$ vertex disjoint paths between u and v, then G cannot be tracked with k trackers.*

Proof Towards a contradiction assume that there exists a set of $k+4$ vertex disjoint paths, $\mathcal{P} = \{P_1, \ldots, P_{k+4}\}$, between u and v, and G can be tracked with k trackers. Let G' be the subgraph induced by $\mathcal{P}$. As each edge and vertex of G is part of some s–t path, there exists an s–t path, say λ, that intersects with G'. Let a and b be the first and last vertices respectively visited by λ in G' (see Fig. 8). We denote the subpaths from s to a and b to t by λ_1 and λ_2, respectively. Here it is possible that, $V(\lambda_1) = \{s\} = \{a\}$ and/or $V(\lambda_2) = \{t\} = \{b\}$. Observe that either there exists a path from a to u that is vertex disjoint from the path from v to b, or there exists a path from a to v that is vertex disjoint from the path from u to b, and either pair of paths can intersect with at most two paths from $\mathcal{P}$. Without loss of generality, assume that there exist mutually disjoint paths between a and u, say $\lambda_3 \subseteq P_1$, and between v and b, say $\lambda_4 \subseteq P_{k+4}$, as shown in Fig. 8. By the pigeonhole principle, if we use at most k trackers, among the paths in $\mathcal{P} \backslash \{P_1, P_{k+4}\}$, there exist two paths that do not contain any trackers. Without loss of generality let us assume that P_2 and P_3 do not contain any trackers, while P_i contains a tracker, for $4 \leq i \leq k+3$. Consider the following two paths:

$$\lambda' = \lambda_1 \cdot \lambda_3 \cdot P_2 \cdot \lambda_4 \cdot \lambda_2$$
$$\lambda'' = \lambda_1 \cdot \lambda_3 \cdot P_3 \cdot \lambda_4 \cdot \lambda_2$$

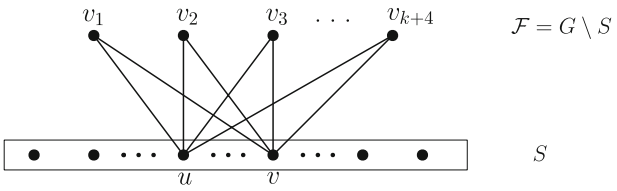


Fig. 9 $k + 4$ vertices in $\mathcal{F}$ connected to the same pair of vertices in S

Observe that λ' and λ'' are two s–t paths with the same sequence of trackers. This contradicts our assumption that G can be tracked by at most k trackers. Hence the result holds.

Note that if $\lambda_3 = P_1$ and $\lambda_4 = P_{k+4}$ i.e. $a = v$ and $b = u$, then the number of vertex disjoint paths between u and v will be greater than or equal to $k + 1$, thus requiring at least k trackers. Also, if a and b lie on the same path in $\mathcal{P}$, then the number of vertex disjoint paths between u and v will be greater than or equal to $k + 2$, thus requiring at least $k + 1$ trackers. The lemma now follows from Corollary 3. □

We use the above lemma in upcoming sections to bound the number of vertices in $\mathcal{F}$ as a function of k. We start by giving a bound for the number of vertices in $\mathcal{F}$ that have at least two neighbors in S.

3.1 Bounding the Number of Vertices in $\mathcal{F}$ with at Least Two Neighbors in S

Lemma 9 implies the following reduction rule.

Reduction Rule 3 *If there exist two vertices in S that have $k + 4$ common neighbors in $\mathcal{F}$, then we return that the given instance is a NO instance.*

Lemma 10 *Reduction Rule 3 is safe and can be implemented in polynomial time.*

Proof Consider the case when there exist $k+4$ vertices in $\mathcal{F}$ that are neighbors of two distinct vertices u and v in S (see Fig. 9). Note that this leads to the formation of $k+4$ vertex disjoint paths between u and v. We know from Lemma 9 that in such a case, G cannot be tracked with at most k trackers. This proves safeness of the reduction rule. To implement the rule, we can consider each pair of vertices ($\mathcal{O}(k^2)$ pairs) in S, and compare their neighbors in $\mathcal{F}$ ($\mathcal{O}(n^2)$ comparisons). Hence the rule can be implemented in polynomial time. □

Corollary 4 *If Reduction Rule 3 is not applicable, then the number of vertices in $\mathcal{F}$ that have at least two neighbors in S is at most $2k^2(k + 3)$.*

Proof Note that when a pair of vertices, say u, v, in S, is adjacent to a vertex, say x, in $\mathcal{F}$, then x has at least two neighbors in S. There are at most $\binom{2k}{2}$ pairs in S, and after application of Reduction Rule 3, each pair can be adjacent to at most $k + 3$ vertices in $\mathcal{F}$. Hence, there can be at most $2k^2(k + 3)$ vertices in $\mathcal{F}$ that have at least two neighbors in S. □

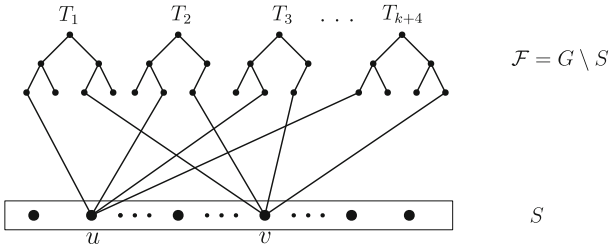


Fig. 10 $k + 4$ trees in $\mathcal{F}$ connected to same pair of vertices in S

3.2 Bounding the Number of Trees in $\mathcal{F}$

Now we explain how the argument for Reduction Rule 3 can be also used to bound the number of trees in $\mathcal{F}$ that are adjacent to at least two vertices in S.

Reduction Rule 4 *If there exist two vertices in S and there exist $k + 4$ trees in $\mathcal{F}$ such that both the vertices in S have at least one neighbor in each of the $k + 4$ trees, then we return that the given instance is a NO instance.*

Lemma 11 *Reduction Rule 4 is safe and can be implemented in polynomial time.*

Proof Note that there exists a unique path between any two vertices in a tree. So if there exist two vertices, say u and v in S, such that both have at least one neighbor in each of the $k + 4$ trees in $\mathcal{F}$, then $k + 4$ vertex disjoint paths exist between u and v passing through these trees. See Fig. 10. From Lemma 9, we know that in such a case, G cannot be tracked with at most k trackers. Hence safeness of the reduction rule holds. For applying the rule, we can consider each pair of vertices ($\mathcal{O}(k^2)$ pairs) in S, and compare if their neighbors in $\mathcal{F}$ ($\mathcal{O}(n^2)$ comparisons) belong to the same tree ($\mathcal{O}(n)$ time). Hence the rule can be implemented in polynomial time. □

Since there exist at most $\binom{2k}{2}$ pairs in S, and after application of Reduction Rule 4, each pair can be adjacent to at most $k + 3$ trees, we have the following corollary.

Corollary 5 *If Reduction Rule 4 is not applicable, then the number of trees in $\mathcal{F}$ that have at least two neighbors in S is at most $2k^2(k + 3)$.*

Next we bound the number of trees with exactly one neighbor in S. Towards this we first show the following.

Lemma 12 *Any induced subgraph G' of G containing at least one edge will contain a pair of vertices u and v, such that, (a) there exists a path in G from s to u, say P_{su}, and another path from v to t, say P_{vt}, (b) $V(P_{su}) \cap V(P_{vt}) = \emptyset$, (c) $V(P_{su}) \cap V(G') = \{u\}$ and $V(P_{vt}) \cap V(G') = \{v\}$.*

Proof Consider an edge $e = (x, y)$ in $E(G')$. We know that e participates in at least one s–t path in G, say P. We denote the subpaths of P from s to x, and from y to t, as P_{sx} and P_{yt} respectively. Observe that P_{sx} and P_{yt} are vertex disjoint. Let $u \in V(G')$ be the first vertex in P_{sx} encountered in G', while traversing from s to x,

and $v \in V(G')$ be the last vertex encountered in G', while traversing P_{yt} from y to t. Observe that the path from s to u (which is a subpath of P_{sx}) and the path from v to t (which is a subpath of P_{yt}) are vertex disjoint and intersect with G' only at u and v. Hence the claim holds. □

Next, we show that the only trees that have exactly one neighbor in S are the ones that contain either s or t, and hence at most two in count.

Lemma 13 *The number of trees in $\mathcal{F}$ that have a single neighbor in S is at most* 2.

Proof In order to prove the lemma, we will prove that a tree in $\mathcal{F}$ with single neighbor in S necessarily contains either s or t or both s and t, and hence the count of such trees is at most 2. For contradiction let us assume that there exists a tree R in $\mathcal{F}$ that contains neither s nor t, and has a single neighbor in S. First consider the case when R consists of only one vertex, say u. Due to Reduction Rule 1, u participates in at least one s–t path, so u must have at least two neighbors in S. Thus we are left with the possibility where R necessarily contains at least two vertices, and hence, contains an edge. By Lemma 12, there exists a pair of vertices u, v in R, such that there is a path in G from s to u and a path from v to t such that these paths are mutually disjoint, and contain no other vertices from R. This implies that R has at least two neighbors (the neighbors of u and v in those paths) in S. Note that above argument can be extended for the case when R has more than 2 vertices as well. This contradicts the initial assumption, and hence the lemma holds. □

Note that if a tree in $\mathcal{F}$ has zero neighbors in S, then the graph is disconnected. But due to Reduction Rule 1, each vertex participates in at least one s–t path. Hence, each tree has at least one neighbor in S. From Corollary 5, the number of trees with at least two neighbors in S is at most $2k^2(k+3)$, and from Lemma 13, the number of trees with a single neighbor in S is at most two. Thus we have the following corollary.

Corollary 6 *If Reduction Rules 1–4 are not applicable, then the number of trees in $\mathcal{F}$ is at most $2k^2(k+4)$.*

We have bounded the number of trees in $\mathcal{F}$, and the number of vertices in the trees in $\mathcal{F}$ with at least two neighbors (in Sect. 3.1). Now we are left to bound the number of vertices in $\mathcal{F}$ with exactly one neighbor in S. Note that due to Reduction Rule 1, all leaf vertices (except possibly s and t) necessarily have neighbors in S. And, a bound on the number of leaf vertices in $\mathcal{F}$ immediately gives a bound on the number of internal vertices (may or may not have neighbors in S) of the trees in $\mathcal{F}$.

3.3 Bounding the Number of Vertices in $\mathcal{F}$ with Exactly One Neighbor in S

We bound the number of vertices in a single tree in $\mathcal{F}$, that are adjacent to a particular vertex of S. Consider a tree R in forest $\mathcal{F}$, and a vertex $f \in S$. Let N_f be the set of vertices in R that are adjacent to f. In this section we give a bound on the cardinality of N_f. Let R' be a subtree of R with the minimum number of vertices, such that R' contains all vertices of N_f. Note that the leaf vertices of R' are in N_f. Note that if $|N_f| < 3$, then the number of vertices in each tree in $\mathcal{F}$ that are adjacent to a single vertex is 2, and hence, asymptotically same as the number of trees in $\mathcal{F}$.

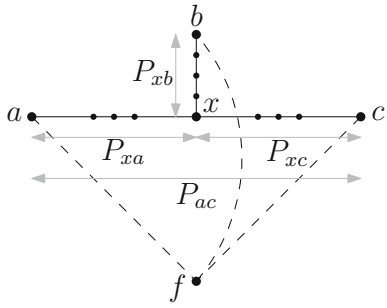


Fig. 11 Illustration of Lemma 14 (solid lines denote edges of the tree R'', dashed lines denote edges between f and its neighbors in R'')

Lemma 14 *Let R'' be a subtree of R' with the minimum number of vertices, such that it contains three vertices from N_f. Then any tracking set T must contain at least one vertex of R''.*

Proof The proof is based on the argument that there will be three vertex disjoint paths formed between some vertex in R'' and f. At most two of these paths may contain s and t, thus requiring the remaining one path to necessarily contain a tracker. Let $\{a, b, c\} \subseteq N_f$, be the three vertices contained in R''. The situation is shown in Fig. 11. Let T be a tracking set which does not contain any vertex from R''. Observe that all the leaf vertices of R'' belong to $\{a, b, c\}$. Furthermore, at least two of a, b or c must be leaf vertices in R''. Without loss of generality assume that a and c are leaves of R''. Consider the path P_{ac} from a to c in R''. Without loss of generality assume that b is connected to P_{ac} via the path P_{xb} (see Fig. 11) joining P_{ac} at vertex x. We denote the paths from x to a and x to c by P_{xa} and P_{xc}, respectively. Note that P_{xb} could be a single vertex, i.e., $b = x$.

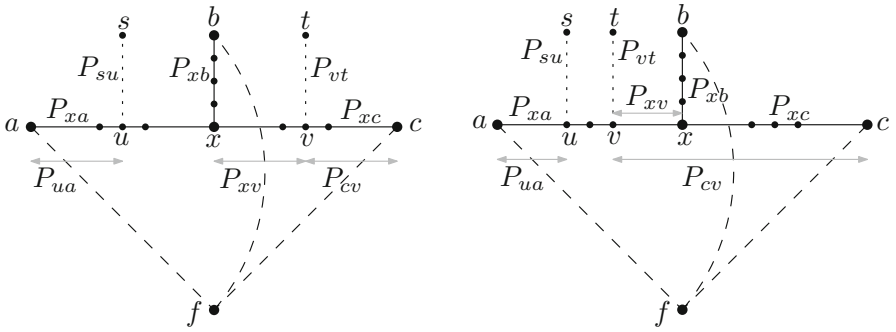
Let G' be the graph induced by $\{f\} \cup V(R'')$. From Lemma 12 we know that there exists a pair of vertices $u, v \in V(G')$, such that there exists a path in G from s to u, say P_{su}, and a path from v to t, say P_{vt}, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$, and both these paths do not contain any vertices from G' except u and v. See Fig. 12a. Depending on the locations of u and v in G', we consider the following three cases.

Case 1 (u and v lie on different paths among P_{xa}, P_{xb} and P_{xc}): Without loss of generality assume $u \in V(P_{xa})$, $v \in V(P_{xc})$, other cases can be proved similarly. See Fig. 12a. Observe that $P_{su} \cdot P_{ua} \cdot f \cdot P_{xb} \cdot P_{xv} \cdot P_{vt}$ and $P_{su} \cdot P_{ua} \cdot f \cdot P_{cv} \cdot P_{vt}$ contain the same sequence of trackers (f may or may not contain a tracker). This contradicts the assumption that T is a tracking set.

Case 2 (u and v lie on same path among P_{xa}, P_{xb} and P_{xc}): Without loss of generality assume $u, v \in P_{xa}$. See Fig. 12b. Observe that $P_{su} \cdot P_{ua} \cdot f \cdot P_{cv} \cdot P_{vt}$ and $P_{su} \cdot P_{ua} \cdot f \cdot P_{xb} \cdot P_{xv} \cdot P_{vt}$ contain the same sequence of trackers. This contradicts the assumption that T is a tracking set.

Case 3 ($u = f$ or $v = f$): Without loss of generality assume that $u = f$ and $v \in V(R'')$. Observe that there exist two paths from f to v in G'. In particular, $P_{su} \cdot P_{av} \cdot P_{vt}$ and $P_{su} \cdot P_{bv} \cdot P_{vt}$ are the two paths that contain the same sequence of trackers, hence, contradicting the assumption that T is a tracking set.

□



(a) Case 1 **(b)** Case 2

Fig. 12 Illustration of Lemma 14 (solid lines denote edges of the tree R'', dashed lines denote edges between f and its neighbors in R'', dotted lines denote the paths between R'' and s and t)

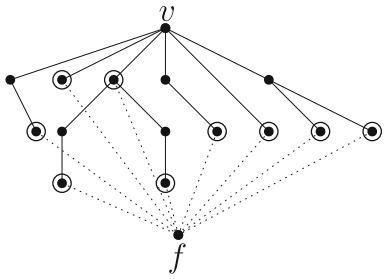


Fig. 13 Degree of each vertex in R' is at most $k+4$ (circled vertices belong to N_f)

Next we use the above lemma to bound the degree of vertices in R'.

Lemma 15 *Let v be a vertex in R'. If v has more than $k+4$ neighbors in R', then it is a NO instance.*

Proof Observe that if there exists a vertex v, with degree at least $k+4$ in R', then there exist at least $k+3$ leaves in R' (one neighbor of v might be its parent in R'). Hence there are at least $k+4$ vertex disjoint paths between v and f, either all passing through descendants of v in R', or $k+3$ of them passing through descendants of v, while the remaining one path passes through an ancestor of v. See Fig. 13. From Lemma 9 we know that in such a case, G cannot be tracked with at most k trackers. Hence the lemma holds. □

Since R' is a minimum tree containing N_f, we have that all the leaf vertices of R' belong to N_f. Next we have the following lemma.

Lemma 16 *Let $p \geq 1$ be an integer. If the number of vertices in N_f is at least $(2k+9)p$ then we can partition R' into at least p disjoint subtrees, each containing at least three vertices from N_f.*

Proof We prove the claim by induction on p. Consider the base case when $p = 1$. If the number of vertices in N_f is at least $2k+9$ then clearly R' is a single tree containing

at least three vertices from N_f. Suppose the claim holds for any value $p < q$. Next we prove that the statement holds for $p = q$. Let R' contains $(2k+9)q$ vertices from N_f. Let w be a vertex closest to a leaf vertex, such that the subtree rooted at w contains at least three vertices from N_f. We denote the subtree rooted at w by T_w. Due to Lemma 15, it is known that the degree of w is upper bounded by $k+4$. Hence, w can have at most $k+4$ children, the subtree rooted at each having at most two vertices from N_f. Thus the number of vertices of N_f present in the subtrees rooted at the children of w is at most $2k+8$. Note that w may also belong to N_f. Hence, if we remove the subtree rooted at w from R', it will still contain $(2k+9)(q-1)$ vertices from N_f. We know that $R' \setminus T_w$ contains at least $(2k+9)(q-1)$ vertices from N_f. Hence, from induction hypothesis, we can partition $R' \setminus T_w$ into $q-1$ disjoint subtrees, each containing three vertices from N_f. Hence, the claim holds. □

From Lemma 14 we know that each partition mentioned in Lemma 16 needs a separate tracker. This proves the safeness of the following reduction rule.

Reduction Rule 5 *If there exists a vertex in S that is adjacent to at least $(2k+9)(k+1)$ vertices of a tree in $\mathcal{F}$, we return that the given instance is a NO instance.*

Note that the above rule can be implemented in polynomial time by considering each vertex in S, and counting its neighbors in each tree in $\mathcal{F}$.

Corollary 7 *If Reduction Rule 5 is not applicable, then the total number of vertices from each tree in $\mathcal{F}$ that are adjacent to a vertex in S is at most $2k(2k+9)(k+1)$.*

Proof If the bound given in the corollary does not hold, then there will be a vertex in S that will have more than $(2k+9)(k+1)$ vertices in a tree and Reduction Rule 5 will apply. □

3.4 Wrapping up: Polynomial Kernel and FPT Algorithm

From Corollaries 5 and 7, we have the following corollary.

Corollary 8 *The number of vertices in a tree in $\mathcal{F}$, that have at least one neighbor in S, is at most $2k^2(k+4) + 2k(2k+9)(k+1) = 2k(3k^2+19k+8)$.*

Now we are left to bound the number of vertices in $\mathcal{F}$ that have no neighbors in S. Note that every leaf vertex in $\mathcal{F}$ has at least one neighbor in S (due to Reduction Rule 1). So in order to bound the number of vertices in $\mathcal{F}$ with no neighbors in S, we need to bound the number of internal (non-leaf) vertices in the trees in $\mathcal{F}$. Next we give a reduction rule that will help to bound the number of internal vertices in each tree.

Reduction Rule 6 *If there exist three vertices, say a, b and c, each of degree two, such that $(a,b), (b,c) \in E(G)$, then delete vertex b, and introduce the edge (a,c) in $E(G)$.*

Lemma 17 *Reduction Rule 6 is safe and can be implemented in polynomial time.*

Proof Note that, due to Reduction Rule 1, there does not exist an edge between vertices a and c, as otherwise a, b, and c would form an isolated triangle, and hence, none of them will be part of an s–t path. Now observe that by removing the vertex b, and introducing the edge (a, c), the only change in the structure of the graph is that the s–t paths passing through b will have their lengths reduced by one vertex. Hence, proving that there exists an optimal tracking set that does not contain b suffices to prove the lemma. Let T be an optimal tracking set that contains b. Observe that any s–t path that contains either of the three vertices, say b, must also contain the other two vertices a and c. Thus a single vertex (any one) amongst a, b, c is sufficient to be included in T to indicate containment of vertices in an s–t path, if required. Two vertices from a, b, c have to be included in T if their mutual sequence is different in two s–t paths having the same vertex sets.

First, we consider the case when T does not contain a and c, but contains b. Observe that if an s–t path P contains b, then the relative ordering between b and some other vertex, say d, in path P is same as the relative ordering between c and d in P. Hence $(T \setminus \{b\}) \cup \{c\}$ is also a tracking set.

Next, we consider the case when T contains both a and b. Since the relative ordering of a and b is the same as that of a and c, we can replace b by c in this case. Thus $(T \setminus \{b\}) \cup \{c\}$ is also a tracking set. The same argument also holds if T contains both b and c, in this case an alternate tracking set being $(T \setminus \{b\}) \cup \{a\}$. Hence the reduction rule is safe. This rule can be implemented in polynomial time by considering all $\binom{n}{3}$ vertex sets of size three in G, and checking if a set of three vertices, each of degree two, forms an induced path in G. □

Lemma 18 *If none of the reduction rules are applicable, then the number of vertices in a tree in $\mathcal{F}$ that do not have any neighbors in S is at most* $10k(3k^2 + 19k + 8)$.

Proof In a tree, we denote the set of leaf vertices by V_1, the set of vertices with degree two by V_2, and the set of vertices with degree three or more by V_3. Since G is preprocessed by Reduction Rule 1, each vertex participates in at least one s–t path, so there cannot be any vertices with degree one in G, except for s and t. Thus each leaf vertex in a tree in $\mathcal{F}$ necessarily has a neighbor in S. Thus, by Corollary 8, the number of vertices in V_1 is upper bounded by $2k(3k^2 + 19k + 8)$. Observe that both V_2 and V_3 belong to the set of internal vertices in a tree. First we consider the tree to be consisting of only V_1 and V_3, giving a bound on these two. By standard graph theoretic properties of a tree, we know that the number of internal vertices in a tree with degree≥ 3 are upper bounded by the number of leaves in the tree. Hence, $|V_3| \leq |V_1|$. Hence the number of vertices in V_3 is upper bounded by $2k(3k^2 + 19k + 8)$. Due to Reduction Rule 6, we know that an induced path consisting of only degree two vertices can contain at most two vertices. Hence in any tree in $\mathcal{F}$, between every pair of closest vertices from $V_1 \cup V_3$ there can be an induced path consisting of at most two vertices from V_2. Hence the number of vertices in V_2 is upper bounded by $2(|V_1| + |V_3|)$, i.e. $|V_2| \leq 8k(3k^2 + 19k + 8)$. Hence $|V_2| + |V_3| \leq 2k(3k^2 + 19k + 8)$. Thus the overall upper bound on the number of vertices in $\mathcal{F}$ that do not have any neighbors in S is $10k(3k^2 + 19k + 8)$. □

From Corollary 8 and Lemma 18, we have the following corollary.

Corollary 9 *If none of the reduction rules are applicable, then the number of vertices in each tree in $\mathcal{F}$ is at most* $12k(3k^2 + 19k + 8)$.

Proof of Theorem 2 From Corollaries 6 and 9, in case of a YES instance, the total number of vertices in $\mathcal{F}$ is at most $12k(3k^2 + 19k + 8)2k^2(k + 4)$, and the total number of vertices in G is at most $12k(3k^2 + 19k + 8)2k^2(k + 4) + 2k$. Thus the total number of vertices in G is at most $72k^6 + 744k^5 + 1920k^4 + 768k^3 + 2k$ if it is a YES instance.

Since $\mathcal{F}$ is a forest, the total number of edges in $\mathcal{F}$ is at most $12k(3k^2 + 14k + 8)2k^2(k+4) - 1$. From Reduction Rule 5, we know that the total number of vertices in each tree in $\mathcal{F}$, that are adjacent to a single vertex in S is at most $2k(2k+8)(k+1)$. From Corollary 6, we know that the number of trees in $\mathcal{F}$ is at most $2k^2(k+4)$. Thus the total number of edges between $\mathcal{F}$ and S is at most $2k(2k+8)(k+1)2k^2(k+4)2k$. The total number of edges among the vertices in S is at most $4k^2$. Thus the total number of edges in G is at most $12k(3k^2+19k+8)2k^2(k+4)-1+2k(2k+8)(k+1)2k^2(k+4)2k+4k^2$, which is $\mathcal{O}(k^7)$. □

Theorem 4 TRACKING PATHS *is FPT when parameterized by the solution size, and the running time of the FPT algorithm is* $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$.

Proof First we apply Theorem 1, to get an equivalent instance (kernel) with $\mathcal{O}(k^6)$ vertices. Then we run through all vertex subsets of size k in the kernel, and use Algorithm 1 to check if a particular subset is a tracking set for G, or not. Using the proof of Theorem 2, we can find all subsets of size k in $\mathcal{O}(k^{6k})$ time. From Algorithm 1, we can verify if a subset of k vertices is a tracking set for G in polynomial time. Thus we have a $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ time FPT algorithm for TRACKING PATHS. □

4 Conclusions

In this paper we have shown that the TRACKING PATHS problem is NP-complete. We also show the problem to be fixed-parameter tractable by proving the existence of a polynomial sized kernel. This is achieved by exploiting the connection between a feedback vertex set and a tracking set. An open problem is to improve the size of the kernel, and the running time of the FPT algorithm for the problem. Other directions to explore are to find approximation algorithms, to study the problem for other graph classes like planar graphs and directed graphs, and to consider weighted versions of the problem.

References

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discrete Math. **12**(3), 289–297 (1999)
2. Banik, A., Katz, M.J., Packer, E., Simakov, M.: Tracking paths. In: Algorithms and Complexity—10th International Conference, CIAC 2017, pp. 67–79 (2017)
3. Bhatti, S., Xu, J.: Survey of target tracking protocols using wireless sensor network. In: 2009 Fifth International Conference on Wireless and Mobile Communications, pp. 110–115 (2009)

4. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms, 1st edn. Springer, Berlin (2015)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
6. Gupta, R., Das, S.R.: Tracking moving targets in a smart sensor network. In: IEEE 58th Vehicular Technology Conference, vol. 5, pp. 3035–3039 (2003)
7. Harary, F., Melter, R.A.: On the metric dimension of a graph. Ars. Comb. **2**(191–195), 1 (1976)
8. Hernando, C., Mora, M., Pelayo, I.M., Seara, C., Wood, D.R.: Extremal graph theory for metric dimension and diameter. Electron. J. Comb. **17**(1), R30 (2010)
9. Karp, R.M.: Reducibility Among Combinatorial Problems. Springer, Berlin (1972)
10. Kawarabayashi, K., Kobayashi, Y., Reed, B.: The disjoint paths problem in quadratic time. J. Comb. Theory Ser. B **102**(2), 424–435 (2012)
11. Khuller, S., Raghavachari, B., Rosenfeld, A.: Landmarks in graphs. Discrete Appl. Math. **70**(3), 217–229 (1996)
12. Kociumaka, T., Pilipczuk, M.: Faster deterministic feedback vertex set. Inf. Process. Lett. **114**(10), 556 (2014)
13. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Comput. Surv. **39**(1), 3 (2007)
14. Slater, P.J.: Leaves of trees. Congr. Numer. **14**(549–559), 37 (1975)
15. Snoeren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Kent, S.T., Strayer, W.T.: Hash-based ip traceback. SIGCOMM Comput. Commun. Rev. **31**(4), 3–14 (2001)
16. Thomassé, S.: A $4k^2$ kernel for feedback vertex set. ACM Trans. Algorithms (TALG) **6**(2), 32 (2010)

Affiliations

Aritra Banik[1] · Pratibha Choudhary[2] · Daniel Lokshtanov[3] · Venkatesh Raman[4,5] · Saket Saurabh[4,5]

Aritra Banik
aritra@niser.ac.in

Daniel Lokshtanov
daniello@ucsb.edu

Venkatesh Raman
vraman@imsc.res.in

Saket Saurabh
saket@imsc.res.in

1 National Institute of Science Education and Research Bhubaneswar, Bhubaneswar, India

2 Indian Institute of Technology Jodhpur, Jodhpur, India

3 University of California Santa Barbara, Santa Barbara, USA

4 The Institute of Mathematical Sciences, HBNI, Chennai, India

5 UMI ReLaX, Chennai, India