Finding an induced path that is not a shortest path

Eli Berger\(^1\)
University of Haifa

Paul Seymour\(^2\)
Princeton University, Princeton, NJ 08544

Sophie Spirkl\(^3\)
Rutgers University, Piscataway, NJ 08854

July 18, 2019; revised March 9, 2021

\(^1\)Supported by Israel Science Foundation Grant 100004639 and Binational Science Foundation USA-Israel Grant 100005728.

\(^2\)Supported by AFOSR grant A9550-19-1-0187 and NSF grant DMS-1800053.

\(^3\)Current address: University of Waterloo, Waterloo, Ontario, Canada N2L3G1. This material is based upon work supported by the National Science Foundation under Award No. DMS-1802201.
Abstract

We give a polynomial-time algorithm that, with input a graph $G$ and two vertices $u, v$ of $G$, decides whether there is an induced $uv$-path that is longer than the shortest $uv$-path.
1 Introduction

All graphs in this paper are finite and simple. For a graph $G$ and $u, v \in V(G)$, the $G$-distance $d_G(u, v)$ (or $d(u, v)$ when there is no danger of confusion) is the number of edges in a shortest $uv$-path in $G$; let $d(u, v) = \infty$ if there is no such path. Let $P$ be an induced $uv$-path. The length of $P$ is the number of edges of $P$. We call $P$ a non-shortest $uv$-path (uv-NSP) if the length of $P$ is more than $d(u, v)$.

Given a graph $G$ and $u, v \in V(G)$, how can we test whether there are two induced $uv$-paths of different lengths, or equivalently, whether there is a uv-NSP? Deciding this in polynomial time is surprisingly non-trivial. (It is important that we want induced paths; if we just want paths of different lengths, the question is much easier.) Our main result is the following:

1.1. **There is an algorithm that, given a graph $G$ and $u, v \in V(G)$, decides whether there is a uv-NSP in time $O(|G|^{18})$.**

This will be proved in section 2. We use a dynamic programming algorithm for one step of the proof, and this technique gives a class of further results that we develop in section 3. In particular, in that section we will prove:

1.2. **For fixed $k$, there is a polynomial-time algorithm that, given a graph $G$ and $u, v \in V(G)$, decides whether there is an induced path between $u$ and $v$ in $G$ of length exactly $d(u, v) + k$.**

Many variants of finding induced paths and pairs of induced paths have been considered previously; for instance

1.3 (Bienstock [1]). **The following problems are NP-hard:**

- Given $u, v \in V(G)$, decide whether there is an induced $uv$-path of odd (even) length.
- Given nonadjacent $u, v \in V(G)$, decide whether there are two induced $uv$-paths $P_1$ and $P_2$ with no edges between $V(P_1) \setminus \{u, v\}$ and $V(P_2) \setminus \{u, v\}$ (that is, decide whether $u, v$ lie in an induced cycle).

We have two more NP-hardness results, that are new as far as we know (we omit the proofs). The first is:

1.4. **The following problem is NP-hard:**

- **Input:** A graph $G$ and $u, v \in V(G)$.
- **Output:** “Yes” if there exist two induced $uv$-paths $P$ and $Q$ such that there are no edges between $V(P) \setminus \{u, v\}$ and $V(Q) \setminus \{u, v\}$, and $P$ is a shortest $uv$-path; and “No” otherwise.

This is in contrast with 3.4, which implies that the problem is polynomial-time solvable if both $P$ and $Q$ are required to be shortest paths (or at most a fixed constant amount longer than a shortest path). In view of 1.1, it is natural to ask:

1.5. **For fixed $k > 1$, is there a polynomial-time algorithm that, given a graph $G$ and $u, v \in V(G)$, decides whether there is an induced $uv$-path $P$ in $G$ of length at least $d(u, v) + k$?”
This remains open, even for $k = 3$ (the algorithm of this paper does the case $k = 1$, and can be adjusted to do the case $k = 2$). It is necessary to fix $k$, because of the following, our second NP-hardness result (again, we omit its proof):

1.6. The following problem is NP-hard:

- **Input:** A graph $G$ and $u, v \in V(G)$.
- **Output:** “Yes” if there exists a $uv$-NSP of length at least $2d_G(u, v)$ and “No” if there is no such path.

2 Finding an induced non-shortest path

In this section, we prove 1.1. We start with some definitions. Let $G$ be a graph, and $u, v \in V(G)$. A vertex $x \in V(G)$ is $uv$-straight if $d(u, x) + d(x, v) = d(u, v)$. Let $F$ be the set of $uv$-straight vertices. For $i \in \{0, \ldots, d(u, v)\}$, let $V_i = \{x \in F : d(u, x) = i\}$; we call $V_i$ the $uv$-layer of height $i$, and we say its elements have height $i$; and we call the sequence $V_0, \ldots, V_d(u,v)$ the $uv$-layering of $G$. It follows that for $i, j \in \{0, \ldots, d(u, v)\}$ with $|i - j| \geq 2$, there are no edges between $V_i$ and $V_j$, and moreover, for $i \in \{1, \ldots, d(u, v) - 1\}$, every vertex in $V_i$ has a neighbour in $V_{i-1}$ and in $V_{i+1}$.

We call a path $Q$ with $V(Q) \subseteq F$ monotone (leaving the dependence on $u, v$ to be understood) if $|V(Q) \cap V_i| \leq 1$ for all $i \in \{0, \ldots, d(u, v)\}$ (and therefore $Q$ is induced); and it follows that the vertices of $Q$ are in $|V(Q)|$ $uv$-layers of consecutive heights. For every vertex $x \in F$, there is a monotone $ux$-path intersecting precisely $V_0, \ldots, V_{d(u,x)}$ and a monotone $vx$-path intersecting precisely $V_{d(u,x)}, \ldots, V_{d(u,v)}$, and from the definition of $uv$-monotonicity, it follows that both of these paths are shortest paths. If $K \subseteq V(G)$, $N(K)$ or $N_G(K)$ denotes the set of all vertices in $V(G) \setminus K$ that have a neighbour in $K$.

We need the following simple “dynamic programming” algorithm (this method is further developed in section 3).

2.1. There is an algorithm with the following specifications:

- **Input:** A graph $G$, and vertices $u, v$ of $G$ such that every vertex of $G$ is $uv$-straight, and the $uv$-layering $V_0, \ldots, V_{d(u,v)}$ of $G$; also $h, k$ with $0 \leq h < k \leq d(u, v)$, and four vertices $s_1, t_1, s_2, t_2$ of $G$, where $s_1, s_2 \in V_h$ and $t_1, t_2 \in V_k$.

- **Output:** Two monotone paths $P_1, P_2$ with ends $s_1, t_1$ and $s_2, t_2$, respectively, such that $V(P_1 \cap P_2) = \emptyset$ and there are no edges between $V(P_1)$ and $V(P_2)$, or a determination that no such paths exist.

- **Running time:** $O(|G|^4)$.

**Proof.** We may assume that $s_1 \neq s_2$ and $s_1, s_2$ are nonadjacent, because otherwise the paths do not exist. Let $C_i$ be the ordered pair $(s_1, t_1)$. For $i = h + 1 \leq k$, compute the set $C_i$ of all pairs $x, y \in V_i$ such that $x, y$ are distinct and nonadjacent, and for some $p, q \in C_{i-1}$, $px$ and $qy$ are edges and $py, qx$ are not edges. Check whether $(t_1, t_2) \in C_k$; if so output that the desired paths exist, and otherwise output that they do not exist. It is easy to see correctness of the algorithm; and its running time is $O(|G|^4)$ (to see the last, note that each quadruple $(p, q, x, y)$ is examined only once). This proves 2.1.

\[\]
Conveniently, in order to solve 1.1 it is enough to handle the case when all vertices are \( uv \)-straight, because of the next result.

2.2. There is an algorithm with the following specifications:

- **Input:** A graph \( G \) and \( u, v \in V(G) \).
- **Output:** Either a \( uv \)-NSP, or a graph \( G' \) with \( u, v \in V(G') \subseteq V(G) \) such that \( G' \) has a \( uv \)-NSP if and only if \( G \) has a \( uv \)-NSP, and such that every vertex of \( G' \) is \( uv \)-straight in \( G' \).
- **Running time:** \( O(|G|^3) \).

**Proof.** Let \( G \) be a graph, and \( u, v \in V(G) \). We will give an algorithm, with running time \( O(|G|^2) \), that outputs either

- a \( uv \)-NSP in \( G \); or
- a determination that every vertex of \( G \) is \( uv \)-straight; or
- a graph \( H \) with \( V(H) \) a proper subset of \( V(G) \) and with \( u, v \in V(H) \), such that \( H \) has a \( uv \)-NSP if and only if only if \( G \) has a \( uv \)-NSP.

Here is the algorithm. First, compute the set \( F \) of \( uv \)-straight vertices of \( G \), and the \( uv \)-layering \( V_0, \ldots, V_{d(u,v)} \) of \( G \). If \( F = V(G) \), we output that every vertex of \( G \) is \( uv \)-straight, and stop. Hence we may assume that \( V(G) \setminus F \neq \emptyset \).

Compute the vertex set \( K \) of a connected component of \( G \setminus F \). (This takes time \( O(|G|^2) \).) Test whether \( N(K) \) contains non-adjacent vertices \( x, y \) with \( d(u, x) < d(u, y) \); if so, in this case we will output a \( uv \)-NSP in \( G \), as follows. Choose \( x, y \in N(K) \) such that \( d(u, y) − d(u, x) \) is maximum. Let \( i = d(u, x) \) and \( j = d(u, y) \). It follows that no vertex in \( V_0, \ldots, V_{i−1} \) has a neighbour in \( K \) (for otherwise such a vertex contradicts the choice of \( x \)); and similarly, no vertex in \( V_{j+1}, \ldots, V_{d(u,v)} \) has a neighbour in \( K \). Now let \( P_1 \) be a monotone \( ux \)-path, let \( P_2 \) be a monotone \( yv \)-path, and let \( Q \) be an induced \( xy \)-path of \( G \) with interior in \( K \). (We can find these paths in time \( O(|G|^2) \).) It follows that the concatenation \( P_1 \cdot Q \cdot P_2 \) is an induced \( uv \)-path; and since \( V(Q) \cap K \neq \emptyset \) (because \( x, y \) are nonadjacent), it follows from the definition of \( K \) and \( F \) that \( P_1 \cdot Q \cdot P_2 \) is a \( uv \)-NSP. Output this and stop.

Thus we may assume that there are no such \( x, y \). In this case we will output \( H \) as in the third bullet above. It follows that there do not exist \( i, j \) with \( 1 \leq i, j \leq d(u, v) \) and \( j \geq i+2 \), such that both \( V_i, V_j \) have nonempty intersection with \( N(K) \) (because then \( x \in V_i \cap N(K) \) and \( y \in V_j \cap N(K) \) would be nonadjacent); and consequently \( N(K) \) is contained in \( V_i \cup V_{i+1} \) for some \( i \in \{0, \ldots, d(u, v) − 1\} \), and \( N(K) \cap V_i \) is complete to \( N(K) \cap V_{i+1} \). Let \( H \) be obtained from \( G \) by deleting \( K \) and adding edges to make \( N(K) \) a clique. We output \( H \) and stop.

To prove correctness, we must show that \( H \) has a \( uv \)-NSP if and only if \( G \) does. Suppose first that \( P \) is a \( uv \)-NSP of \( G \). Since \( N(K) \) is a clique of \( H \), there is a \( uv \)-path of \( H \) with vertex set a subset of \( V(P) \); let \( Q \) be the shortest such path. We claim that \( Q \) is a \( uv \)-NSP of \( H \). If \( V(P) = V(Q) \), this follows from the choice of \( P \). Otherwise, \( Q \) contains an edge \( e \in E(H) \setminus E(G) \). Since \( e \) connects two vertices at the same distance from \( u \), it follows that every induced \( uv \)-path containing \( e \) is a \( uv \)-NSP of \( H \), as claimed, and so \( H \) has a \( uv \)-NSP.
Now suppose that \(Q\) is a \(uv\)-NSP of \(H\). If \(Q\) does not contain an edge in \(E(H) \setminus E(G)\), then \(Q\) is a \(uv\)-NSP of \(G\), so we assume that \(Q\) contains such an edge. Since \(N(K)\) is a clique of \(H\), it follows that \(Q\) contains exactly two vertices \(x, y \in N(K)\), and \(xy \not\in E(G)\). Let \(P\) be obtained from \(Q\) by replacing \(xy\) by an induced \(xy\)-path with interior in \(K\). Then \(P\) is a \(uv\)-NSP of \(G\), since \(P\) contains a vertex of \(K\). This proves that \(H\) has a \(uv\)-NSP if and only if \(G\) does, and so completes the proof of correctness of the algorithm. The running time is \(O(|G|^2)\).

Let us call the algorithm just described “algorithm A”. For an algorithm as specified in 2.2, first apply algorithm A to \(G\). If its first or second output applies, we are done, so we may assume that its third output applies, that is, it outputs a graph \(H\). We proceed as follows. Enumerate all \((G)\) apply algorithm A to vertex \(x\) of correctness of the algorithm. The running time is \(O(|G|^3)\). This proves 2.2.

If there is a \(uv\)-NSP in \(G\), there is a shortest \(uv\)-NSP, and this has some convenient properties that will help us detect a \(uv\)-NSP. We have:

2.3. Let \(G\) be a graph and let \(u, v \in V(G)\), such that every vertex of \(G\) is \(uv\)-straight. For each vertex \(x\), let \(h(x)\) be its height. Let \(P\) be a shortest \(uv\)-NSP in \(G\) (assuming that one exists). Let \(P_u\) be the longest monotone subpath of \(P\) containing \(u\), and let \(P_v\) be the longest monotone subpath of \(P\) containing \(v\). Let \(s\) denote the endpoint of \(P_u\) that is not \(u\), and let \(t\) denote the endpoint of \(P_v\) that is not \(v\). Then \(P_u\) and \(P_v\) are disjoint, and \(h(x) \leq h(s)\) for every \(x \in V(P) \setminus V(P_v)\), and \(h(x) \geq h(t)\) for every \(x \in V(P) \setminus V(P_u)\). Consequently \(h(s) \geq h(t)\).

**Proof.** Since \(P\) is not monotone, it follows that \(P_u\) and \(P_v\) are disjoint. Let \(x \in V(P) \setminus V(P_v)\) be chosen with \(h(x)\) maximum, breaking ties by choosing the vertex closest to \(u\) along \(P\). Let \(Q\) be a monotone \(ux\)-path, and let \(P'\) be the subpath of \(P\) from \(u\) to \(x\). Let \(Q'\) denote the concatenation of \(P'\) and \(Q\). We claim that \(Q'\) is shorter than \(P\). This follows since the subpath of \(P\) from \(x\) to \(v\) is not monotone (because \(x \not\in V(P_v)\)), and the subpath of \(Q'\) from \(x\) to \(v\) is monotone. Since \(P\) is a shortest \(uv\)-NSP, it follows that \(Q'\) is not a \(uv\)-NSP, and hence \(Q'\) is monotone. In particular, \(P'\) is monotone, and so \(V(P') \subseteq V(P_u)\). From the choice of \(x\), it follows that \(P' = P_u\); and so \(x = s\). From the choice of \(x\), and from the symmetry between \(u\) and \(v\), this proves the first statement of 2.3. The second, that \(h(s) \geq h(t)\), follows immediately since \(P\) is not monotone.

With notation as in 2.3, we define \(h(s) - h(t)\) to be the **twist** of \(P\). Thus the twist is non-negative.

2.4. For each integer \(k \geq 0\), there is an algorithm with the following specifications:

- **Input:** A graph \(G\) and \(u, v \in V(G)\) such that every vertex of \(G\) is \(uv\)-straight.

- **Output:** A \(uv\)-NSP in \(G\), or a determination that there is no shortest \(uv\)-NSP in \(G\) with twist exactly \(k\).

- **Running time:** \(O(|G|^{k+6})\).

**Proof.** We proceed as follows. Enumerate all \((k + 4)\)-tuples \((x, y, v_1, \ldots, v_{k+2})\) of vertices of \(G\) with the following properties:
• $x, y$ are adjacent, and $h(y) = h(x) + 1$;

• $v_1 \cdots v_{k+2}$ is a $(k + 2)$-vertex path with $h(v_i) = h(y) + i - 1$ for $1 \leq i \leq k + 2$; and

• $v_1$ is nonadjacent to $x$, and $v_i$ is nonadjacent to $x, y$ for $2 \leq i \leq k + 2$.

For all such choices of $(x, y, v_1, \ldots, v_{k+2})$, we proceed as follows:

• Compute a monotone path $Q_u$ from $u$ to $x$, and a monotone path $Q_v$ from $v_{k+2}$ to $v$.

• Compute the graph $H$ obtained from $G$ by deleting all vertices and neighbours of $V(Q_u) \cup V(Q_v) \cup \{x\} \cup \{v_2, \ldots, v_{k+2}\}$ except for $y$ and $v_1$.

• Test whether $H$ contains an induced path $Q$ from $v_1$ to $y$. If so, return the concatenated path $Q' = uQ_u \cdot x \cdot yQ_v \cdot v_1 \cdot v_2 \cdots v_{k+2} \cdot Q_v \cdot v$ and stop.

If all choices of $(k + 4)$-tuples have been examined and no path has been returned, we report that there is no shortest $uv$-NSP with twist exactly $k$, and stop.

To prove correctness, we must show that if the algorithm returns a path then this path is a $uv$-NSP; and if it returns no path, then there is no shortest $uv$-NSP with twist exactly $k$. Thus, suppose that the algorithm returns a path $Q'$. From the construction of $H$, it follows that $Q'$ is an induced path. Moreover, since $Q'$ contains $v_1$ and $y$, and since $h(v_1) = h(y)$, it follows that $Q'$ is a $uv$-NSP.

Now we show that if there is a shortest $uv$-NSP with twist exactly $k$, then the algorithm returns a path for one of the choices of $x, y, v_1, \ldots, v_{k+2}$. Let $P$ be a shortest $uv$-NSP with twist exactly $k$, and define $P_u, P_v, s, t$ as before. Choose $x, y \in V(P_u)$ with $h(y) = h(t)$ and $h(x) = h(t) - 1$ (thus $x, y$ are adjacent). Let $v_1 \cdots v_{k+2}$ be a subpath of $P_v$ with $v_1 = t$. Then $(x, y, v_1, \ldots, v_{k+2})$ is one of the $(k + 4)$-tuples to which the algorithm is applied, and we claim that for this application, the algorithm returns a path.

Let $H$ be as in the algorithm. We claim that the subpath $P'$ of $P$ from $v_1$ to $y$ is contained in $H$. Since $h(x) = h(t) - 1$ and so every vertex $z$ in $V(Q_u) \setminus \{x\}$ satisfies $h(z) \leq h(t) - 2$, it follows from (1) that $z$ has no neighbours in $P'$. Similarly, no vertex in $V(Q_v)$ has a neighbour in $P'$. Since $x, y \in V(P_u)$, it follows that the only neighbour of $x$ in $P'$ is $y$. Since $v_1, \ldots, v_{k+2} \in V(P_v)$, it follows that the only possible neighbour of $v_2, \ldots, v_{k+2}$ in $P'$ is $v_1$. This proves our claim. Since $P'$ is a path from $v_1$ to $y$ in $H$, it follows that the algorithm returns a path $Q'$. This proves correctness of the algorithm. Since it is easy to check the running time, this proves 2.4.

2.5. There is an algorithm with the following specifications:

• **Input:** A graph $G$ and $u, v \in V(G)$ such that every vertex of $G$ is $uv$-straight.
- **Output**: A $uv$-NSP in $G$, or a determination that there is no shortest $uv$-NSP in $G$ with twist at least six.

- **Running time**: $O(|G|^{18})$.

**Proof.** Enumerate all 14-tuples $(s_0, s_1, \ldots, s_6, t_1, \ldots, t_6, t_7)$ of vertices of $G$ with the following properties:

- $s_0, s_1, \ldots, s_6, t_1, \ldots, t_6, t_7$ are all distinct;
- $s_0$-$s_1$-$s_2$-$s_3$, $s_4$-$s_5$-$s_6$, $t_1$-$t_2$-$t_3$, and $t_4$-$t_5$-$t_6$-$t_7$ are paths;
- $h(s_i) = h(t_i)$ for $1 \leq i \leq 6$;
- $h(s_0) + 3 = h(t_1) + 2 = h(t_2) + 1 = h(t_3) \leq h(t_4) = h(t_5) - 1 = h(t_6) - 2 = h(t_7) - 3$; and
- $s_i$ is non-adjacent to $t_j$ for all $i \in \{0, \ldots, 6\}$ and $j \in \{1, \ldots, 7\}$.

For each such 14-tuple $(s_0, s_1, \ldots, s_6, t_1, \ldots, t_6, t_7)$, run the following algorithm:

- Compute a monotone path $Q_u$ from $u$ to $s_0$, and a monotone path $Q_v$ from $t_7$ to $v$.
- Using 2.1, compute (in time $O(|G|^4)$) a pair $R_u$, $R_v$ of monotone paths such that $R_u$ is an $s_3$s$_4$-path, $R_v$ is a $t_3$t$_4$-path, and there are no edges between $V(R_u)$ and $V(R_v)$; or if no such pair of paths exists, move on to the next 14-tuple.
- Let $P'_u$ and $P'_v$ be respectively the concatenations
  
  $$w$-Q$_u$-$s_0$-$s_1$-$s_2$-$s_3$-$R_u$-$s_4$-$s_5$-$s_6$
  
  $$t_1$-$t_2$-$t_3$-$R_v$-$t_4$-$t_5$-$t_6$-$t_7$-$Q_v$-$v$.

  Compute the graph $H$ obtained from $G$ by deleting all vertices of $P'_u \setminus \{s_6\}$ and all their neighbours except $s_6$, and deleting all vertices of $P'_v \setminus \{t_1\}$ and all their neighbours except $t_1$.
  Test whether there is an induced path $Q$ from $t_1$ to $s_6$ in $H$, and if so, return the concatenated path $w$-$P'_u$-$s_6$-$Q$-$t_1$-$P'_v$-$v$ and stop.

If the algorithm runs through all the 14-tuples without returning a path, return that there is no shortest $uv$-NSP in $G$ with twist at least six, and stop.

This completes the description of the algorithm. To prove correctness, we must show that if the algorithm returns a path, then it is a $uv$-NSP, and otherwise that there is no shortest $uv$-NSP in $G$ with twist at least six.

If the algorithm returns a path $Q'$, then the construction implies that $Q'$ is an induced path; and since $Q'$ contains $s_1$, $t_1$ with $h(s_1) = h(t_1)$, it follows that $Q'$ is a $uv$-NSP. It remains to show that if a shortest $uv$-NSP $P$ exists with $h(s) - h(t) \geq 6$ in the usual notation, then the algorithm returns a path for some choice of the 14-tuple. Consider the 14-tuple such that $s_6 = s$, and $t_1 = t$, $\{s_0, \ldots, s_6\} \subseteq V(P_u)$, and $\{t_1, \ldots, t_7\} \subseteq V(P_v)$. This 14-tuple exists since $h(s) - h(t) \geq 6$, and so there are at least six vertices in $P_u$ that each have the same height as some vertex in $P_v$.

Now we need to show that, when applied to this 14-tuple, the algorithm above returns a path. Let $P'$ be the subpath of $P$ from $s$ to $t$. It follows from (1) that there are no edges from $V(Q_u)$ or
Since the analysis of running time, which is straightforward; so this proves 2.5.

above does indeed return a path. This completes the proof of correctness of the algorithm. We omit

3 Dynamic programming

The dynamic programming technique used in 2.1 can be extended, and in this section we develop that. A path forest means a graph in which every component is a path (possibly of length zero); and a path forest in G means an induced subgraph of G that is a path forest. (Thus it consists of a set of induced paths of G, pairwise vertex-disjoint and with no edges of G joining them.)

Let $V_1, \ldots, V_n$ be pairwise disjoint subsets of $V(G)$, with union $V(G)$, such that for all $i, j \in \{1, \ldots, n\}$, if $j \geq i + 2$ then there are no edges between $V_i$ and $V_j$. We call $(V_1, \ldots, V_n)$ an altitude. We are given a graph G and an altitude $(V_1, \ldots, V_n)$ in G, and we will show how to test whether there is a path forest in G with certain properties, that contains only a bounded number of vertices from each $V_i$.

Let $X \subseteq V(G)$, and let $H, H'$ be path forests in G. We say they are $X$-equivalent if

- $V(H) \cap X = V(H') \cap X$;
• $H$, $H'$ have the same number of components; and

• for each component $P$ of $H$, there is a component $P'$ of $H'$ with the same ends and same length as $P$.

This is an equivalence relation.

Again, let $X \subseteq V(G)$. A path forest $H$ is $h$-restricted in $G$ relative to $X$ if $|V(H) \cap X| \leq h$, and there are at most $h$ components of $H$ that have no end in $X$. Now let $(V_1, \ldots, V_n)$ be an altitude in $G$. A path forest $H$ is $h$-narrow (with respect to $(V_1, \ldots, V_n)$) if for $1 \leq i \leq n$, $H[V_i \cup \cdots \cup V_n]$ is $h$-restricted in $G[V_i \cup \cdots \cup V_n]$ with respect to $V_i$.

Let $1 \leq i \leq n$. Let $C_i$ be the set of all equivalence classes, under $V_i$-equivalence, that contain a path forest in $G[V_i \cup \cdots \cup V_n]$ that is $h$-narrow with respect to $(V_i, \ldots, V_n)$. Algorithmically, we may describe $C_i$ by explicitly storing such a path forest.

We observe:

3.1. If $h$ is fixed, with $G, V_1, \ldots, V_n$ as above, for $1 \leq i < n$ we can compute $C_i$ from a knowledge of $C_{i+1}$ in polynomial time.

Proof. There are only polynomially many equivalence classes in $C_{i+1}$. (This is where we use the condition that at most $h$ components of $H$ have no end in $X$, in the definition of “$h$-restricted”.) For each one, take a representative member $H'$ say. There are only polynomially many induced subgraphs $J$ of the graph $G[V_i \cup V_{i+1}]$ such that $V(J) \cap V_{i+1} = V(H') \cap V_{i+1}$ and $|V(J) \cap V_i| \leq h$. For each such $J$, check whether $H' \cup J$ is $h$-narrow in $G[V_i \cup \cdots \cup V_n]$ with respect to $(V_1, \ldots, V_n)$, and if so record its equivalence class under $V_i$-equivalence. To see that every member of $C_i$ is recorded, observe that if $H$ is a path forest in $G[V_i \cup \cdots \cup V_n]$ that is $h$-narrow with respect to $(V_1, \ldots, V_n)$, then $H \setminus V_i$ is a path forest in $G[V_{i+1} \cup \cdots \cup V_n]$ that is $h$-narrow with respect to $(V_{i+1}, \ldots, V_n)$; and if $H'$ is another member of the equivalence class in $C_{i+1}$ that contains $H \setminus V_i$, then its union with $J = H[V_i \cup V_{i+1}]$ is $h$-narrow with respect to $(V_1, \ldots, V_n)$ and $V_i$-equivalent to $H$. This proves 3.1.

We deduce:

3.2. For all fixed $h \geq k \geq 0$, there is a polynomial-time algorithm that, given pairs $(s_1, t_1), \ldots, (s_r, t_r)$ of a graph $G$, and integers $n_1, \ldots, n_r \geq 0$, and an altitude $(V_1, \ldots, V_n)$ in $G$, computes whether there is a path forest in $G$, $h$-restricted with respect to $(V_1, \ldots, V_n)$, with $r$ components, where the $i$th component has ends $s_i, t_i$ and has length $n_i$.

Proof. First compute $C_n$; then $n - 1$ applications of 3.1 allow us to compute $C_1$, and from $C_1$ we can read off the answer.
Similarly, by trying all possibilities for \( n_1, \ldots, n_r \), we obtain a generalization of 2.1:

**3.4.** For fixed \( h \) and \( r \), there is a polynomial-time algorithm with the following specifications, where \( V_i \) is the set of vertices with distance exactly \( i \) from \( v \):

- **Input:** A graph \( G \), a vertex \( v \in V(G) \) and \( r \) pairs \((s_1, t_1), \ldots, (s_r, t_r) \in V(G)\).

- **Output:** A path forest \( H \) of \( G \) with \( r \) components \( P_1, \ldots, P_r \), such that for each \( i \), \( P_i \) has ends \( s_i, t_i \) and \( |V(H) \cap V_j| \leq h \) for all \( j \in \mathbb{N} \), or a determination that no such path forest exists.

**Acknowledgments**

The first author was supported by Israel Science Foundation Grant 100004639 and Binational Science Foundation USA-Israel Grant 100005728. The second author was supported by AFOSR grant A9550-19-1-0187 and NSF grant DMS-1800053. This material is based upon work supported by the National Science Foundation under Award No. DMS-1802201 (Spirkl).

**References**