On the Complexity of Secluded Path Problems

Tesshu Hanaka
Kyushu University, Fukuoka, Japan

Daisuke Tsuru
Kyushu University, Fukuoka, Japan

Abstract

This paper investigates the complexity of finding secluded paths in graphs. We focus on the Short Secluded Path problem and a natural new variant we introduce, Shortest Secluded Path. Formally, given an undirected graph $G = (V, E)$, two vertices $s, t \in V$, and two integers k, l, the Short Secluded Path problem asks whether there exists an s-t path of length at most k with at most l neighbors. This problem is known to be computationally hard: it is W[1]-hard when parameterized by the path length k or by cliquewidth, and para-NP-complete when parameterized by the number l of neighbors. The fixed-parameter tractability is known for $k + l$ or treewidth. In this paper, we expand the parameterized complexity landscape by designing (1) an XP algorithm parameterized by cliquewidth and (2) fixed-parameter algorithms parameterized by neighborhood diversity and twin cover number, respectively. As a byproduct, our results also provide parameterized algorithms for the classic s-t k-Path problem. Furthermore, we introduce the Shortest Secluded Path problem, which seeks a shortest s-t path with the minimum number of neighbors. In contrast to the hardness of the original problem, we reveal that this variant is solvable in polynomial time on unweighted graphs. We complete this by showing that for edge-weighted graphs, the problem becomes W[1]-hard yet remains in XP when parameterized by the shortest path distance between s and t.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Secluded path, Parameterized complexity, Polynomial-time algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.4

Funding *Tesshu Hanaka*: JSPS KAKENHI Grant Numbers JP21K17707, JP22H00513, JP25K03077, and JST, CRONOS, Japan Grant Number JPMJCS24K2.

1 Introduction

Path-finding problems are one of the most fundamental graph problems. These problems have been well studied due to their high practicality [1, 3, 7, 9]. However, in many real-world scenarios, simply finding the shortest path is not sufficient. For instance, for secure communication, one may seek a data transmission path for sensitive information that minimizes exposure to potential eavesdroppers. Similarly, in a transportation network, a convoy path that avoids potential attackers could lead to more reliable and secure traveling. In robotics, a path for a robot might need to avoid areas with high sensor noise or potential collisions. These scenarios highlight the need for path-finding problems that consider the *seclusion* of the path from undesirable elements.

Motivated by these applications, the Short Secluded Path problem has been well studied [2, 11, 21, 28]. A vertex subset $U \subseteq V$ is called l-*secluded* if it has at most l neighbors. Formally, Short Secluded Path is defined as follows.

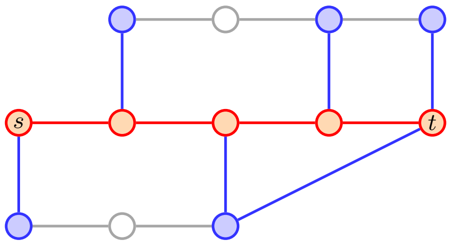

Figure 1 An illustration of a 5-secluded s-t path P of length 5. The s-t path P consists of the red vertices. The blue vertices are neighbors of P.

SHORT SECLUDED PATH

Instance: An undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, two vertices $s, t \in V$, and two integers k, l.

Goal: Determine whether G has an l-secluded s-t path of length at most k.

Figure 1 is an illustration of an l-secluded s-t path of length at most k when $l = 5$ and $k = 5$.[1] Unfortunately, SHORT SECLUDED PATH is NP-complete in general since it is equivalent to s-t HAMILTONIAN PATH if we set $k = n$ and $l = 0$. Due to the NP-hardness of s-t HAMILTONIAN PATH on restricted graph classes [6, 16], SHORT SECLUDED PATH is NP-hard even on planar bipartite graphs of maximum degree 3, strongly chordal split graphs, and chordal bipartite graphs.

Thus, there has been considerable interest in its parameterized complexity. For the natural parameters k and l, Luckow and Fluschnik [21] show that SHORT SECLUDED PATH is W[1]-hard when parameterized by k, whereas it is fixed-parameter tractable (FPT) when parameterized by $k + l$. Note that the problem is para-NP-complete when parameterized by l due to the hardness of s-t HAMILTONIAN PATH. In [28], van Bevern et al. study the fixed-parameter tractability and kernelization complexity of SHORT SECLUDED PATH for structural parameters related to tree-like structures such as treewidth, vertex cover number, feedback vertex set number, and feedback edge set number. For treewidth tw, the authors presented an FPT algorithm for SHORT SECLUDED PATH that runs in $2^{O(\mathrm{tw})}n^{O(1)}$ time. For kernelization, the problem admits a polynomial kernel when parameterized by the combination of k, l, and feedback vertex set number, and also when parameterized by feedback edge set number alone. In contrast, it does not admit a polynomial kernel when parameterized by vertex cover number, by the combination of l and feedback vertex set number, or by the combination of k, l, and treewidth.

1.1 Our contribution

In this paper, we present parameterized algorithms for SHORT SECLUDED PATH with respect to structural parameters related to dense structures such as clique-width, twin cover number, and neighborhood diversity. Our results lead to a more comprehensive understanding of the parameterized complexity of SHORT SECLUDED PATH under structural graph parameters (see Figure 2).

First, we propose an XP algorithm when parameterized by clique-width. Since s-t HAMILTONIAN PATH is known to be W[1]-hard when parameterized by clique-width [12, 16], this XP algorithm provides a tight complexity bound for this parameter. Furthermore,

[1] In this paper, the length of a path is defined as the number of vertices in the path.

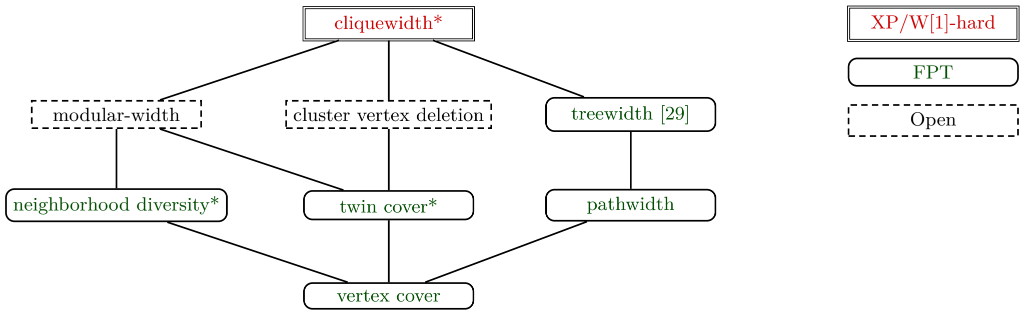

Figure 2 Parameterized complexity of Short Secluded Path with respect to structural graph parameters. The connection between two parameters indicates that the upper parameter p is bounded by some computable function $f(\cdot)$ of the lower parameter q; that is, $p \le f(q)$. Parameters marked with an asterisk $(*)$ represent our contributions in this paper. Double-bordered rectangles indicate that the parameterized problem belongs to XP but is W[1]-hard; rounded rectangles indicate that the parameterized problem is fixed-parameter tractable (FPT); and dotted rectangles represent cases that remain open.

we design a $\mathrm{nd}(G)^{O(\mathrm{nd}(G)^2)}n^{O(1)}$-time algorithm and a $2^{O(\mathrm{tc}(G)^2)}n^{O(1)}$-time algorithm for Short Secluded Path when parameterized by neighborhood diversity $\mathrm{nd}(G)$ and twin cover number $\mathrm{tc}(G)$, respectively.

Here, it is worth mentioning that our algorithms are designed for Secluded k-Path, which asks whether G has an l-secluded s-t path of length *exactly* k. Thus, by setting $l = n$, our results provide parameterized algorithms for the classical graph problem s-t k-Path. To the best of our knowledge, the parameterized complexity of s-t k-Path with respect to clique-width, twin cover number, and neighborhood diversity was unsettled.

Then we consider the Shortest Secluded Path problem. Unlike Short Secluded Path, the goal of this problem is to find a minimally secluded s-t path among all shortest s-t paths. Formally, Shortest Secluded Path is defined as follows.

Shortest Secluded Path

Instance: An undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, and two vertices $s, t \in V$.

Goal: Find a shortest s-t path P that minimizes the size of its open neighborhood $|N(V(P))|$ in G.

Interestingly, we show that Shortest Secluded Path is solvable in polynomial time on unweighted graphs. This is in contrast to the hardness of Short Secluded Path. However, for positive integer edge-weighted graphs, the problem becomes W[1]-hard when parameterized by the shortest path distance d between s and t. To complement this, we finally present an XP algorithm when parameterized by d.

1.2 Related work

There has been extensive literature on secluded subgraph problems. The study of this area is initiated by Chechik et al. [2], who first introduced the notion of *seclusion* for connectivity problems on graphs. Their original formulation, which they term *exposure*, is defined as the size of the closed neighborhood of a vertex subset. In their seminal work, Chechik et al. consider two problems: Secluded Path and its generalization Secluded Steiner Tree, where the goal is to find a path or a Steiner tree that minimizes its exposure. The authors show that while Secluded Path is hard to approximate, it is solvable in polynomial time on graphs with bounded degree. For Secluded Steiner Tree, they present a fixed-parameter algorithm parameterized by treewidth. Subsequently, Fomin et al. show that Secluded Steiner Tree is also FPT when parameterized by the exposure of a solution.

A significant shift in perspective came from van Bevern et al. [27], who decouple the solution size from the exposure, and introduce the notion of *l-secludedness*. Within this framework, they investigate the parameterized complexity for finding secluded versions of several fundamental substructures on graphs, such as separators, dominating sets, $\mathcal{F}$-free vertex deletion sets, and independent sets. Building on this work, Golovach et al. [15] show that Connected Secluded $\mathcal{F}$-Free Subgraph is FPT when parameterized by l, and Donkers et al. [8] present a faster FPT algorithm for Secluded Induced Tree parameterized by l. Recently, the concept of seclusion has been extended further, with Mallek et al. [22] studying secluded subgraph problems on directed graphs.

2 Preliminaries

In this paper, we use standard graph-theoretic notations. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. We let $n = |V|$ and $m = |E|$. For a positive integer n, let $[n] := \{1, 2, \cdots, n\}$

For a vertex $v \in V$, its neighborhood is $N_G(v) = \{u \in V \mid \{v, u\} \in E\}$. For a vertex subset $U \subseteq V$, $N_G(U) = \{v \in V \setminus U \mid u \in U, \{u, v\} \in E\}$ denotes the set of neighbors of U. For simplicity, we sometimes use $N(v)$ and $N(U)$ instead of $N_G(v)$ and $N_G(U)$. The subgraph induced by U is denoted by $G[U]$.

The length of a path is defined by the number of vertices on the path. For two vertices $u, v \in V$, the shortest path distance between u and v, denoted by $\mathrm{dist}(u, v)$, is defined as the number of edges (resp., the sum of the weights of edges) in a shortest u-v path on unweighted graphs (resp., edge-weighted graphs). Two distinct vertices u and v are called *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. In particular, twins u and v are called *true twins* if the edge $\{u, v\}$ exists, and otherwise called *false twins*.

We assume that the readers are familiar with the basic notions of parameterized complexity [5].

2.1 Cliquewidth

In this subsection, we define the cliquewidth of a graph G. The definition relies on the concept of a k-labeled graph, which we introduce first.

▶ **Definition 1** (k-labeled graph)**.** *Let k be a positive integer. A k-labeled graph is a pair $(G, \mathtt{lab}_G)$ of a graph G and a function $\mathtt{lab}_G : V \to \{1, \cdots, k\}$.*

▶ **Definition 2** (Cliquewidth)**.** *The cliquewidth of a graph G, denoted by $\mathrm{cw}(G)$, is the minimum integer k such that a k-labeled graph $(G, \mathtt{lab}_G)$ can be constructed by repeatedly applying the following operations.*

(O1) *Add a new vertex v with label $i \in [k]$.*

(O2) *Take the disjoint union $(G \oplus H, \mathtt{lab}_{G\oplus H})$ of two k-labeled graphs $(G, \mathtt{lab}_G)$ and $(H, \mathtt{lab}_H)$, with*

$$\mathtt{lab}_{G\oplus H}(v) = \begin{cases} \mathtt{lab}_G(v) & \text{if } v \in V(G), \\ \mathtt{lab}_H(v) & \text{otherwise.} \end{cases}$$

(O3) *Take distinct labels $i, j \in [k]$ for a k-labeled graph G, and add an edge between every pair of vertices labeled by i and by j.*

(O4) *Take distinct labels $i, j \in [k]$ for a k-labeled graph G, and relabel the vertices of label i to label j.*

This construction process for a k-labeled graph can be represented by a rooted binary tree, called a *k-expression tree*. Each node in this tree corresponds to one of the four operations: an *introduce node* (O1), a *union node* (O2), a *join node* (O3), or a *relabel node* (O4). The leaves of a k-expression tree are always introduce nodes, and conversely, all introduce nodes are leaves. The graph associated with any node is the one constructed by the operations in its subtree, and thus the root of the tree represents the final graph G.

A k-expression tree is *irredundant* if for each edge $\{u, v\} \in E(G)$, there is exactly one corresponding join node that adds this edge. Any k-expression tree can be transformed into an irredundant one with $O(n)$ nodes in linear time [4]. Therefore, we can assume without loss of generality that any given k-expression tree is irredundant.

While computing the exact cliquewidth and an optimal expression tree is NP-hard, a polynomial-time approximation algorithm exists. Specifically, a $(2^{\mathrm{cw}(G)+1} - 1)$-expression tree for a graph G with cliquewidth $\mathrm{cw}(G)$ can be computed in $O(n^3)$ time [17, 24, 25].

2.2 Neighborhood diversity and twin cover

A partition $\mathcal{M} = \{M_1, \ldots, M_r\}$ of V is called a *twin partition* of G if every M_i is a set of twins. We call M_i a *module* of $\mathcal{M}$. Then the neighborhood diversity of G is defined as follows.

▶ **Definition 3** (Neighborhood diversity). *The* neighborhood diversity $\mathrm{nd}(G)$ *of G is the minimum number of modules among all twin partitions of G.*

We can compute the *neighborhood diversity* $\mathrm{nd}(G)$ of G and its twin partition in linear time [20, 23, 26]. By definition, each module forms either a clique (if it contains true twins) or an independent set (if it contains false twins).

The *quotient graph* corresponding to a twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$ is the graph $Q = (\mathcal{M}, E(\mathcal{M}))$, where $E(\mathcal{M}) = \{\{M_i, M_j\} \mid \exists\{u, v\} \in E,\ u \in M_i,\ v \in M_j\}$. We observe that for any two modules M_i and M_j, either there are no edges between them, or every vertex in M_i is adjacent to every vertex in M_j.

A vertex set $X \subseteq V$ is called a *twin cover* if, for every edge $\{u, v\} \in E$, at least one of the following holds: (i) $u \in X$ or $v \in X$, or (ii) u and v are *true twins* (i.e., adjacent and have identical open neighborhoods). The *twin cover number* of G, denoted $\mathrm{tc}(G)$, is the size of a minimum twin cover of G.

2.3 Integer linear programming

In this subsection, we introduce Integer Linear Programming (ILP) and its fixed-parameter tractability.

▶ **Definition 4** (p-Variable Integer Linear Programming Feasibility (p-ILP)). *Given a matrix $A \in \mathbb{Z}^{m \times p}$ and a vector $b \in \mathbb{Z}^m$, p-Variable Integer Linear Programming Feasibility asks whether there exists a vector $x \in \mathbb{Z}^p$ satisfying $Ax \leq b$.*

It is known that p-ILP is fixed-parameter tractable with respect to the number p of variables.

▶ **Theorem 5** ([13, 18, 19]). *p-Variable Integer Linear Programming Feasibility can be solved in $O(p^{2.5p+o(p)} \cdot L)$ time, where L is the number of bits in the input.*

3 XP Algorithm Parameterized by Cliquewidth

In this section, we design an $n^{2^{O(\mathrm{cw})}}$-time algorithm parameterized by cliquewidth for the problem of determining whether G has an s-t path P of length k with exactly l neighbors. By solving this for each k and l, Secluded k-Path and Short Secluded Path can also be solved in time $n^{2^{O(\mathrm{cw})}}$.

▶ **Theorem 6.** *There is an $n^{2^{O(\mathrm{cw})}}$-time algorithm that, for a graph G of cliquewidth* cw*, determines whether it contains an s-t path of length k with l neighbors.*

Our algorithm is based on dynamic programming on a cliquewidth expression tree of G. It is known that for a graph G with cliquewidth $\mathrm{cw}(G)$, a $(2^{\mathrm{cw}(G)+1}-1)$-expression tree $\mathcal{T}$ with $O(n)$ nodes can be computed in $O(n^3)$ time [17, 24, 25]. For our algorithm, we must distinguish the source s and target t. We achieve this by assigning them two new, unique labels. We can thus assume that we are given an r-expression tree $\mathcal{T}$ with $r \le 2^{\mathrm{cw}(G)+1}+1$ where s is assigned label $r-1$ and t is assigned label r.

For a given expression tree $\mathcal{T}$, we execute dynamic programming in a bottom-up manner, from the leaves to the root. For each node μ in $\mathcal{T}$, let $G_\mu = (V_\mu, E_\mu)$ be the labeled subgraph associated with μ. The DP table at each node μ stores boolean values for states defined by a tuple of parameters. A state corresponds to the properties of a set $\mathcal{P}$ of vertex-disjoint paths within G_μ, denoted by $\mathcal{P}_\mu$. Intuitively, each path in $\mathcal{P}_\mu$ represents a sub-path of a potential solution in G_μ, i.e., an s-t path of length k with l neighbors. Thus, the DP state needs to track the number of vertices of the solution, the number of their neighborhoods, the number of other vertices, and the number of sub-paths with respect to the labels of the endpoints. The information on endpoint labels of sub-paths is used when merging sub-paths at a join node. We use the notation $V(\mathcal{P}_\mu)$ as a shorthand for $\bigcup_{P\in\mathcal{P}_\mu} V(P)$, the set of all vertices in any path in $\mathcal{P}_\mu$. The parameters in our DP state are defined as follows:

- For each label $i \in \{1,\dots,r\}$:
  - x_i^μ: the number of vertices in $V(\mathcal{P}_\mu)$ with label i.
  - y_i^μ: the number of vertices in $N(V(\mathcal{P}_\mu)) \cap V_\mu$ with label i.
  - z_i^μ: the number of vertices with label i in V_μ that belong to neither $V(\mathcal{P}_\mu)$ nor its neighborhood.
- For each pair of labels i, j with $1 \le i \le j \le r$:
  - p_{ij}^μ: the number of paths in $\mathcal{P}_\mu$ having one endpoint with label i and the other with label j.

We call $(x_1^\mu,\dots,x_r^\mu)$ the solution count vector, $(y_1^\mu,\dots,y_r^\mu)$ the neighborhood count vector, $(z_1^\mu,\dots,z_r^\mu)$ the remaining vertex count vector, and $(p_{11}^\mu,\dots,p_{rr}^\mu)$ the path count vector, respectively. The DP entry $\mathrm{DP}_\mu[(x_1^\mu,\dots,x_r^\mu),(y_1^\mu,\dots,y_r^\mu),(z_1^\mu,\dots,z_r^\mu),(p_{11}^\mu,\dots,p_{rr}^\mu)]$ is a boolean value. It is `true` if and only if there exists a set of vertex-disjoint paths $\mathcal{P}_\mu$ in G_μ that satisfies all of the following conditions:

1. For each label i, the number of vertices in $V(\mathcal{P}_\mu)$ with label i is exactly x_i^μ, i.e., $|V(\mathcal{P}_\mu)| = \sum_{i=1}^r x_i^\mu$.
2. For each label i, the number of vertices in the neighborhood $N(V(\mathcal{P}_\mu)) \cap V_\mu$ with label i is exactly y_i^μ, i.e., $|N(V(\mathcal{P}_\mu))| = \sum_{i=1}^r y_i^\mu$.
3. For each label i, the number of vertices with label i in V_μ that belong to neither $V(\mathcal{P}_\mu)$ nor its neighborhood is exactly z_i^μ, i.e., $|V_\mu \setminus (V(\mathcal{P}_\mu) \cup N(V(\mathcal{P}_\mu)))| = \sum_{i=1}^r z_i^\mu$.
4. For each pair of labels (i,j), the number of paths in $\mathcal{P}_\mu$ with endpoints labeled i and j is exactly p_{ij}^μ.

For convenience, if a set $\mathcal{P}_\mu$ of paths is empty, $|V(\mathcal{P}_\mu)| = 0$. In this case, its neighborhood is also empty, and this corresponds to the state where $x_i^\mu = y_i^\mu = p_{ij}^\mu = 0$ for all i, j.

From these definitions, $0 \le x_i^\mu, y_i^\mu, z_i^\mu, p_{ij}^\mu \le n$ for all parameters. Thus, the size of the DP table at each node is bounded by $n^{O(r^2)}$. As initialization, we set all entries of the DP tables to `false`.

The existence of a solution in the original graph G is determined by checking the DP table of the root node, γ. A key observation is that an s-t path of length k with l neighbors exists if and only if there exists a path P whose endpoints are labeled by $r-1$ and r in G_γ satisfying $|V(P)| = k$, $|N(V(P))| = l$, and $|V_\gamma \setminus (V(P) \cup N(V(P)))| = n-k-l$. Note that s and t have the unique labels $r-1$ and r, respectively. Therefore, an s-t path of length k with l neighbors exists if and only if $\mathrm{DP}_\gamma[(x_1^\gamma, y_1^\gamma, z_1^\gamma), \cdots, (x_r^\gamma, y_r^\gamma, z_r^\gamma), (p_{11}^\gamma, \cdots, p_{rr}^\gamma)] = $ `true` at the root node γ such that:

- $x_{r-1}^\gamma = x_r^\gamma = 1$,
- $y_{r-1}^\gamma = y_r^\gamma = 0$,
- $z_{r-1}^\gamma = z_r^\gamma = 0$,
- $\sum_{i=1}^r x_i^\gamma = k$,
- $\sum_{i=1}^r y_i^\gamma = l$,
- $\sum_{i=1}^r z_i^\gamma = n-k-l$,
- $p_{(r-1)r}^\gamma = 1$, and
- $p_{ij}^\gamma = 0$ for $i, j (i \le j)$ except $i = r-1$ and $j = r$.

In the following, we describe the recursive formulas for updating the DP table at each node of the expression tree. The updates for introduce, union, and relabel nodes are relatively straightforward, while the update at a join node is the most challenging part. Intuitively, our DP state tracks, for each label, the number of vertices in the solution, their neighborhoods, and the other vertices, as well as the number of vertex-disjoint paths distinguished by their endpoint labels. For an introduce node, which adds a single labeled vertex, we only need to generate the initial states corresponding to this new vertex, forming a trivial path of length one, or being an isolated vertex not part of the solution. For a union node, which combines two disjoint subgraphs, the new DP table is computed by merging states from its two children. As no edges are added between the subgraphs, paths from each part remain separate. Thus, for any pair of valid states (one from each child), we can compute the resulting state's parameters by simple addition. The update for a relabel node that changes label i to j is also based on addition: for each state, the counts associated with label i are consolidated into the counts for label j.

The main challenge lies with the join node, which adds edges between all vertices of label i and all vertices of label j. These new edges may connect multiple existing paths into longer ones. Therefore, the core of the update is to iterate through all valid states of the child node and, for each state, enumerate all feasible ways that the paths can be connected. Although the number of ways to connect paths can be large, it can be shown that this process can be handled in time $n^{O(r^2)}$. The running time for a join node is determined by iterating through all possible resulting states. Since the size of the DP table is $n^{O(r^2)}$, and we must process each state from the child's table, the update at a join node can be performed in time $n^{O(r^2)}$. As the expression tree has $O(n)$ nodes, the total running time is dominated by the join operations, yielding $n^{O(r^2)} = n^{2^{O(\mathrm{cw}(G))}}$. The details of the algorithm are provided in the appendix.

By applying Theorem 6 for each k, l, we can also solve Secluded k-Path and Short Secluded Path.

▶ **Corollary 7.** *Secluded k-Path and Short Secluded Path can be solved in time $n^{2^{O(\mathrm{cw})}}$.*

4 FPT Algorithm Parameterized by Neighborhood Diversity

In this section, we propose a fixed-parameter algorithm for Short Secluded Path parameterized by neighborhood diversity, which runs in $\mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$ time. To show this, we first provide a fixed-parameter algorithm for s-t k-Path by neighborhood diversity.

4.1 FPT algorithm for s-t k-Path

We give an ILP formula with $O(\mathrm{nd}^2)$ variables for determining whether G has an s-t path of length k passing through every module.

Let $\mathcal{G}$ be the quotient graph with respect to a twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$, where r is the number of modules in the partition of G. Suppose that $M_s = \{s\}$ and $M_t = \{t\}$ for $s, t \in V(G)$ are modules in $\mathcal{M}$, and $\mathcal{G}$ is connected.

We define the ILP instance (P) with variables x_{ij}, x_{ji} (for $\{M_i, M_j\} \in E(\mathcal{G})$), and y_i (for $i \in [r]$) subject to the following constraints.

ILP instance (P)

1. $\sum_{i=1}^{r} y_i = k$

For every $i \in [r]$:

2. a. $\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_i)\}} x_{ij} = \sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_i)\}} x_{ji}$ (if $M_i \neq M_s, M_t$)

b. $\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_s)\}} x_{sj} = 1$

$\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_s)\}} x_{js} = 0$

c. $\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_t)\}} x_{jt} = 1$

$\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_t)\}} x_{tj} = 0$

3. a. $\sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_i)\}} x_{ij} = y_i$ (if M_i is an independent set)

b. $1 \leq \sum_{j\in\{l:M_l\in N_{\mathcal{G}}(M_i)\}} x_{ij} \leq y_i$ (if M_i is a clique)

For every partition of $V(\mathcal{G})$ into vertex sets A and B:

4. $\sum_{1\leq i<j\leq r:\{M_i,M_j\}\in E(\mathcal{G})\wedge|\{M_i,M_j\}\cap A|=1} x_{ij} + x_{ji} \geq 1$

For every variable x_{ij}:

5. $x_{ij} \geq 0$

For every variable y_i:

6. $1 \leq y_i \leq |M_i|$

The core idea is to extend the standard *flow-conservation principle* commonly used in network flow problems. We imagine one unit of flow originating at a source module M_s, being consumed at a sink module M_t, and being conserved at all intermediate modules.

The variables x_{ij} and x_{ji} model the flow moving in either direction between two modules M_i and M_j. They also correspond to the number of arcs from M_i to M_j on a directed path P from s to t. This approach captures the path's traversal, potentially back and forth, through the graph of modules. The integer variables y_i track the number of vertices used within each module M_i that the path visits.

The constraint (1) guarantees that the size of a solution is exactly k. The constraints (2) are the low conservation constraints. Constraint (2.a) ensures that for any intermediate module (not a source or sink), the incoming flow equals the outgoing flow, implying that the number of incoming edges and the number of outgoing edges are equal in the intermediate modules. Constraints (2.b) and (2.c) define the source and sink, respectively: exactly one unit of flow leaves the source module M_s, and exactly one unit of flow enters the sink module M_t. Note that $M_s = \{s\}$ and $M_t = \{t\}$ by assumption.

The constraints (3) govern the relationship between the flow passing through a module and the number of vertices used within it (y_i). The constraint (3.a) implies that every incoming edge immediately goes out in an independent set module M. Thus, the number of incoming edges is equal to the number of used vertices in M. The constraint (3.b) implies that P may pass through several vertices and then go out in a clique module M. Thus, the number of incoming edges is at most the number of used vertices in M.

The constraint (4) is the connectivity condition, which ensures that a solution forms a single path. This disallows invalid solutions, such as a subgraph composed of a path and disjoint cycles.

The constraint (5) is a non-negativity constraint. The constraint (6) ensures that for each module, a solution contains at least one vertex in the module and no more vertices than the size of the module.

▶ **Lemma 8.** *The ILP instance* (P) *is feasible if and only if there is an s-t path in G with k vertices that includes at least one vertex from each module $M_i \in \mathcal{M}$.*

From Lemma 8 and Theorem 5, we obtain the following lemma.

▶ **Lemma 9.** *Given a twin partition with r modules, the problem of determining whether there is an s-t path of length k passing through every module can be solved in $r^{O(r^2)}n^{O(1)}$ time.*

4.2 FPT algorithm for Short Secluded Path

Using Lemma 9, we present an FPT algorithm for Secluded k-Path and Short Secluded Path parameterized by neighborhood diversity. A key observation is that if a path visits a module M, then all vertices from this module not on the path are in the neighborhood of the path. Also, all the vertices in modules adjacent to M are in the neighborhood of the path. Therefore, what we only have to do is to solve the problem of determining whether there is an s-t path of length k passing through every guessed module by using Lemma 9.

▶ **Theorem 10.** *Secluded k-Path parameterized by neighborhood diversity* $\mathrm{nd}(G)$ *can be solved in* $\mathrm{nd}(G)^{O(\mathrm{nd}(G)^2)}n^{O(1)}$ *time.*

Proof. First, we compute a twin partition for G with $\mathrm{nd}(G)$ modules in linear time [20,23,26]. We ensure that s and t are in their own singleton modules by splitting the modules containing them. If $s \in M$, we replace M with $M \setminus \{s\}$ and add a new module $M_s = \{s\}$. We do the same for t. This results in a new twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$ with $r \le \mathrm{nd}(G) + 2$. The algorithm proceeds by guessing which modules $\mathcal{M}' \subseteq \mathcal{M}$ contain the vertices of a potential l-secluded s-t path of length at most k. We must always include M_s and M_t in $\mathcal{M}'$. There are $O(2^{r-2}) = O(2^{\mathrm{nd}})$ possible choices for $\mathcal{M}'$.

Let $r' = |\mathcal{M}'|$. For each guess, we check if there exists a valid solution containing at least one vertex in each module in $\mathcal{M}'$ in the subgraph induced by the vertices of $\mathcal{M}'$. If $|r'| > k$ or the induced subgraph is not connected, we can discard this guess.

Let $\mathcal{N} \subseteq \mathcal{M} \setminus \mathcal{M}'$ be the set of modules adjacent to at least one module in $\mathcal{M}'$. For any s-t path P with $V(P) \subseteq \bigcup_{M \in \mathcal{M}'} M$, its neighborhood $N(V(P))$ consists of two parts: (1) all vertices in the modules of $\mathcal{N}$, and (2) all vertices in modules of $\mathcal{M}'$ that are not on the path P. Therefore, $N(V(P)) = (\bigcup_{M \in \mathcal{N}} M) \cup ((\bigcup_{M \in \mathcal{M}'} M) \setminus V(P))$. The size of the neighborhood is $|N(V(P))| = |\bigcup_{M \in \mathcal{N}} M| + |\bigcup_{M \in \mathcal{M}'} M| - |V(P)|$. Since $|V(P)| = k$, $|N(V(P))|$ is given by $|\bigcup_{M \in \mathcal{N}} M| + |\bigcup_{M \in \mathcal{M}'} M| - k$ for a fixed guess $\mathcal{M}'$ (which fixes $\mathcal{N}$). Thus, if $|N(V(P))| > l$, we discard such a guess.

Finally, we only have to determine whether there exists a path of length k in the graph $G[\bigcup_{M' \in \mathcal{M}'} M']$ that visits every module in $\mathcal{M}'$. By Lemma 9, this can be determined in time $r'^{O(r'^2)} n^{O(1)} = \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$. We repeat this for all $O(2^{\mathrm{nd}})$ guesses of $\mathcal{M}'$. The total runtime is $2^{\mathrm{nd}} \cdot \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)} = \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$. ◀

By applying Theorem 10 to Secluded k'-Path for $2 \le k' \le k$, Short Secluded Path can also be solved in $\mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$ time.

▶ **Corollary 11.** *For a graph G of neighborhood diversity* nd*, Short Secluded Path can be solved in $\mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$ time.*

5 FPT Algorithm Parameterized by Twin Cover Number

In this section, we design a $2^{O(\mathrm{tc}^2)} n^{O(1)}$-time algorithm for Short Secluded Path and Secluded k-Path parameterized by twin cover number.

▶ **Theorem 12.** *Secluded k-Path can be solved in $2^{O(\mathrm{tc}(G)^2)} n^{O(1)}$ time.*

Proof. For any graph G, a twin cover S of size tc can be computed in $O(m + n \cdot \mathrm{tc} + 1.2738^{\mathrm{tc}})$ time [14]. Let $X := S \cup \{s, t\}$ and $\tau := |X| (= \mathrm{tc} + 2)$.

First, we guess the set of edges from $G[X]$ that are part of the secluded path. The number of edges in $G[X]$ is at most $\binom{\tau}{2} \le \tau^2$. Since a path on τ vertices has at most $\tau - 1$ edges, we guess subsets of at most $\tau - 1$ edges as candidates for partial solutions inside of X. The number of guessed subsets is bounded by $\sum_{i=0}^{\tau-1} \binom{\tau^2}{i} = \tau^{O(\tau)}$.

For each guess, we check if the subgraph formed by these edges consists of a collection of disjoint paths, if s and t are part of the subgraph because they must be endpoints of a path, and if the sum of the number of vertices in the guessed disjoint paths is at most k. This clearly can be done in polynomial time. If these conditions are not satisfied, we discard the current guess.

Let $P_1, P_2, \dots, P_d$ be sub-paths within $G[X]$ of a guessed partial solution, where $d \le \tau$. Let $X' = \bigcup_{i=1}^{d} V(P_i)$ be the set of vertices in these paths.

Next, we construct a full s-t path by ordering these sub-paths and connecting them with cliques from $G[V \setminus X]$. First, we guess the orderings of the sub-paths. If s and t belong to the same sub-path P_i, then this must be the entire path. In this case, we check if $d = 1$, $|X'| = |V(P_1)| = k$, and $|N(V(P_1))| \le l$. If so, P_1 is indeed an l-secluded s-t path of length k, and otherwise we safely conclude that the guess is invalid. If s and t are in different sub-paths, we fix the path containing s at the beginning and the one containing t at the end. We then guess the ordering of the remaining $d - 2$ paths. Since $d - 2 \le \tau$, the number of such orderings is $(d-2)! = \tau^{O(\tau)}$.

For a fixed ordering of sub-paths, we need to connect them. Since X is a separator and $G[V \setminus X]$ is a union of disjoint cliques, vertices in exactly one clique connect two sub-paths in $G[X]$.

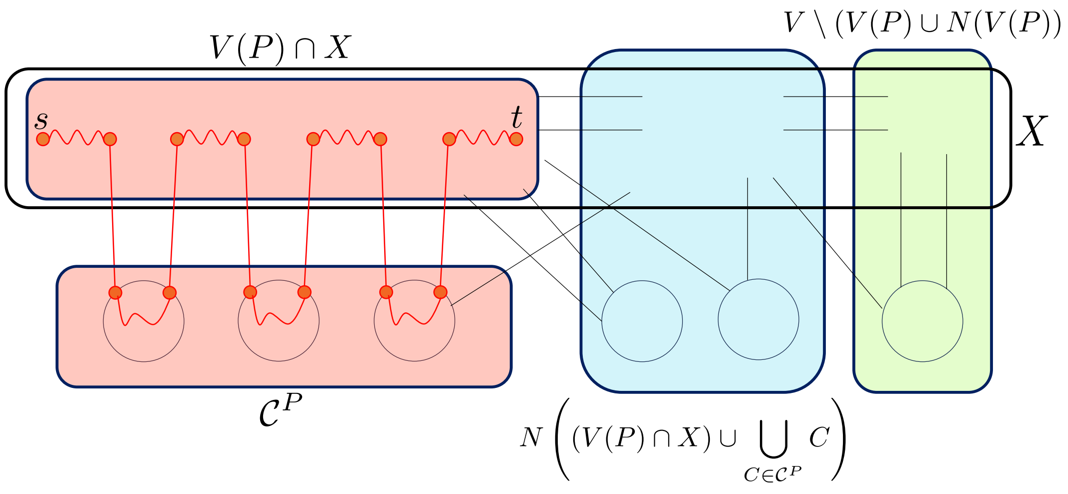
$V(P) \cap X$
$V \setminus (V(P) \cup N(V(P))$
s
t
X
$\mathcal{C}^P$
$N\left((V(P) \cap X) \cup \bigcup_{C \in \mathcal{C}^P} C\right)$

Figure 3 An illustration of a partition of V with respect to a path P in Claim 14.

Here, we categorize cliques in $G[V \setminus X]$ with respect to neighbors in X, and we define the sets of vertices $Y_1, \dots, Y_q$ where vertices in Y_i have the common neighbors in X, that is, $N(u) \cap X = N(v) \cap X$ for $u, v \in Y_i$. Note that $q \leq 2^\tau$. By the definition of a twin cover, each connected component of $G[V \setminus X]$ is a clique and is fully contained in some Y_i. We say that a clique C in $G[V \setminus X]$ is from Y_i if $V(C) \subseteq Y_i$.

To connect the $d-1$ gaps between the ordered sub-paths, we guess which set Y_j provides the connecting clique for each gap. Since there are $d - 1 = O(\tau)$ gaps and $q = O(2^\tau)$ choices for each, the total number of ways to choose the sequence of Y_j's is $q^{d-1} = 2^{O(\tau^2)}$.

For each such guess, we verify whether it is valid. If a chosen Y_j cannot connect the corresponding sub-paths, we reject it. We also calculate the minimum possible path length. Since we must use at least one vertex from a clique in each of the $d-1$ gaps, the length p of an s-t path constructed is at least $p \geq \sum_{i=1}^{d} |V(P_i)| + d - 1$. If $p > k$, we safely reject it. Furthermore, if the number of times a set Y_i is selected is more than $|Y_i|$, then we reject it because we cannot construct an s-t path even by using all the vertices in Y_i.

Now, for each gap between two sub-paths, we must select a specific clique from the chosen Y_i. Then by the following claim, we can focus on at most r largest cliques in each Y_i.

▷ Claim 13. Suppose that Y_i contains r cliques $C_1, \dots, C_r$ ordered by non-increasing size ($|C_1| \geq \ldots \geq |C_r|$). Then if there exists an s-t path P that passes through b cliques from Y_i, there exists an s-t path P' that passes through the b largest cliques from Y_i, $C_1, \dots, C_b$ (or all r cliques if $b \geq r$), such that $|V(P)| = |V(P')|$ and $|N(V(P))| = |N(V(P'))|$.

Now we need to decide vertices to take from the selected cliques. Here, we observe that the vertex set V can be partitioned into four parts: (1) the set $V(P) \cap X$ of vertices on P inside X, (2) the set of vertices $\bigcup_{C \in \mathcal{C}^P} C$ in the selected cliques in $G[V \setminus X]$, (3) the set $N\left((V(P) \cap X) \cup \bigcup_{C \in \mathcal{C}^P} C\right)$ of neighbors of P except for vertices in $\bigcup_{C \in \mathcal{C}^P} C$, and (4) The set $V \setminus V(P) \cup N(V(P))$ of the other vertices which lie outside both X and $N(V(P))$ (see Figure 3). Then the following claim holds.

▷ Claim 14. Let P be an s-t path and let $\mathcal{C}^P$ be the set of cliques in $V \setminus X$ that intersect with $V(P)$. The number of neighbors of the path P is given by:

$$|N(V(P))| = \left|N\left((V(P) \cap X) \cup \bigcup_{C \in \mathcal{C}^P} C\right)\right| + \left|\bigcup_{C \in \mathcal{C}^P} C\right| + |V(P) \cap X| - |V(P)|. \tag{1}$$

If we have fixed the sub-paths in X and the cliques connecting them, we observe that the first three terms $\left|N\left((V(P)\cap X)\cup\bigcup_{C\in\mathcal{C}^P} C\right)\cap V(P)\right|$, $\left|\bigcup_{C\in\mathcal{C}^P} C\right|$, and $|V(P)\cap X|$ in the right-hand side of Equation (1) are determined. Note that $|V(P)\cap X| = |X'| = \sum_{i=1}^{d}|V(P_i)|$. Since $|V(P)| = k$, from Claim 14, $|N(V(P))|$ is also determined.

After fixing the sub-paths in X and the Y_i's for connecting pairs of sub-paths, we can identify the set $\mathcal{C}$ of candidate cliques $\mathcal{C}$ (the largest ones, by Claim 13). Since each clique in Y_i have the same neighbors by the definition of a twin cover, if $|\bigcup_{C\in\mathcal{C}} C| = \sum_{C\in\mathcal{C}}|C| \geq k - |V(P)\cap X|$ is satisfied, we can obtain a path of length k (we just need to arbitrarily choose $k - |V(P)\cap X|$ vertices from selected cliques in $\mathcal{C}$ such that at least one vertex is chosen from each selected clique). This is clearly done in polynomial time.

Finally, we analyze the running time. Guessing the sub-paths in X, the orderings of the sub-paths, and the Y_i's for connecting pairs of sub-paths can be done in time $\tau^{O(\tau)}n^{O(1)}\cdot 2^{O(\tau^2)}n^{O(1)} = 2^{O(\tau^2)}n^{O(1)}$. After the guesses, we can check whether each guess is valid by verifying whether $\sum_{C\in\mathcal{C}}|C| \geq k - |V(P)\cap X|$ in polynomial time. Therefore, the total time is $2^{O(\tau^2)}n^{O(1)} = 2^{O(\mathrm{tc}^2)}n^{O(1)}$. ◀

▶ **Corollary 15.** *Short Secluded Path can be solved in $2^{O(\mathrm{tc}(G)^2)}n^{O(1)}$ time.*

6 Computational Complexity of Shortest Secluded Path

In this section, we study the computational complexity of Shortest Secluded Path.

6.1 Polynomial-time algorithm on unweighted graphs

In this subsection, we present a polynomial-time algorithm for Shortest Secluded Path on unweighted graphs.

Let $k = \mathrm{dist}(s,t)$ be the distance between s and t. For $i \in \{0,1,\ldots,k\}$, we define the layer $L_i = \{v \in V \mid \mathrm{dist}(s,v) = i \text{ and } \mathrm{dist}(v,t) = k-i\}$ as the set of vertices whose distance from s is i and from t is $k-i$. These layers $L_0,\ldots,L_k$ can be computed in linear time by performing a breadth-first search from s and from t, respectively. By definition, $L_0 = \{s\}$ and $L_k = \{t\}$. Let $L = \bigcup_i L_i$ and $R = V\setminus L$. It is easily seen that every vertex v satisfying $\mathrm{dist}(s,v)+\mathrm{dist}(v,t) = k$ is contained in L, and any vertex in R satisfies that $\mathrm{dist}(s,v)+\mathrm{dist}(v,t) > k$. Thus, every shortest s-t path is contained in $G[L]$. Then, the following lemmas hold.

▶ **Lemma 16.** *For any integer $i \in \{0,1,\ldots,k\}$, the neighbors of any vertex in L_i that are also in L must belong to it's layer or adjacent layers. Formally:*

$$N(L_i)\cap L \subseteq L_{i-1}\cup L_{i+1}$$

(Here, we define $L_{-1} = L_{k+1} = \emptyset$.)

Proof. For a vertex $v_i \in L_i$, let $v_j \in L_j (\neq L_i)$ be a neighbor of v_i. By the triangle inequality, we must have $\mathrm{dist}(s,v_i) \leq \mathrm{dist}(s,v_j)+\mathrm{dist}(v_j,v_i)$, which means $i \leq j+1$. Symmetrically, $j \leq i+1$. Together, these imply that j satisfies $|i-j| \leq 1$. ◀

▶ **Lemma 17.** *For any vertex $r \in R$ and any integer $i \in \{0,1,\ldots,k\}$, if r has a neighbor in L_i, then its set of neighbors in L, denoted by $N(r)\cap L$, must satisfy one of the following three conditions: (1) $N(r)\cap L \subseteq L_i$, (2) $N(r)\cap L \subseteq L_{i-1}\cup L_i$, and (3) $N(r)\cap L \subseteq L_i\cup L_{i+1}$.*

From Lemmas 16 and 17, the neighbors of vertices in L_i are in $L_{i-1} \cup L_i \cup L_{i+1} \cup R$. Moreover, the neighbors in R are also adjacent to only $L_{i-1} \cup L_i \cup L_{i+1} \cup R$. This locality is the key to our dynamic programming approach, as it ensures that when extending a path to layer L_{i+1}, the change in the neighborhood of the path only depends on the most recent layers.

Our dynamic programming table, DP, stores values for pairs of adjacent vertices (u_i, u_{i+1}) where $u_i \in L_i$ and $u_{i+1} \in L_{i+1}$. The entry $\mathrm{DP}[u_i, u_{i+1}]$ is defined as the minimum size of the neighborhood of a shortest s-u_{i+1} path that uses the edge $\{u_i, u_{i+1}\}$ as its last edge. If no such path exists, or if $\{u_i, u_{i+1}\} \notin E$, then $\mathrm{DP}[u_i, u_{i+1}] = \infty$. The solution to the overall problem is then $\min_{u_{k-1} \in L_{k-1}} \mathrm{DP}[u_{k-1}, t]$.

For $i = 0$ (the base case), we consider paths of length 2 from s to a vertex $u_1 \in L_1$. Since $\mathrm{dist}(s, u_1) = 1$, the edge $\{s, u_1\}$ must exist. The path consists of edge $\{s, u_1\}$. The size of its neighborhood is $|(N(s) \cup N(u_1)) \setminus \{s, u_1\}|$. Thus, we initialize $\mathrm{DP}[s, u_1] = |N(s) \cup N(u_1)| - 2$.

For $1 \le i < k$, the recursive formula for $\mathrm{DP}[u_i, u_{i+1}]$ is computed as follows:

$$\mathrm{DP}[u_i, u_{i+1}] = \begin{cases} \infty & \text{if } \{u_i, u_{i+1}\} \notin E, \\ \min\limits_{u_{i-1} \in L_{i-1}} \{\mathrm{DP}[u_{i-1}, u_i] + |N(u_{i+1}) \setminus (N(u_{i-1}) \cup N(u_i))|\} & \text{otherwise.} \end{cases}$$

Here, $|N(u_{i+1}) \setminus (N(u_{i-1}) \cup N(u_i))|$ represents the number of new neighbors added when extending the path from u_i to u_{i+1}. Due to the locality shown by Lemmas 16 and 17, u_{i+1} never shares neighbors of $\bigcup_{j=0}^{i-2} L_j$. Thus, these new neighbors are precisely those in $N(u_{i+1})$ that are not already neighbors of the path ending in $\{u_{i-1}, u_i\}$.

Finally, we analyze the running time. The sets L_i can be computed in $O(m + n)$ time. The size of the DP table is $O(n^2)$ since the number of layers is at most n. For three vertices u, v, w, $|N(u) \cap (N(v) \cup N(w))|$ can be computed in polynomial time in advance. Thus, each entry $\mathrm{DP}[u_i, u_{i+1}]$ can be computed in $O(n)$ time from $\mathrm{DP}[u_{i-1}, u_i]$. Therefore, the total running time is $O(n^3)$.

▶ **Theorem 18.** *Shortest Secluded Path can be solved in $O(n^3)$ time.*

6.2 Shortest Secluded Path on weighted graphs

In this subsection, we discuss Shortest Secluded Path on edge-weighted graphs. We assume that each edge weight is a positive integer. We first show Shortest Secluded Path on weighted graphs is W[1]-hard with respect to the shortest path distance between s and t by a reduction from Multicolored Clique. Here, the shortest path distance between s and t is defined by the minimum weight of an s-t path.

Multicolored Clique

Instance: A graph $G = (V_1 \cup \cdots \cup V_k, E)$ with n vertices where each V_i forms an independent set.

Goal: Determine whether G has a clique of size k.

Multicolored Clique is W[1]-hard when parameterized by k even on regular graphs [10].

▶ **Theorem 19.** *Shortest Secluded Path on edge-weighted graphs is W[1]-hard parameterized by the shortest path distance d between s and t in the input graph.*

Proof. We show the parameterized reduction from Multicolored Clique to Shortest Secluded Path.

Let an r-regular graph $G = (V, E)$ be an instance of Multicolored Clique, where $V = V_1 \cup V_2 \cup \cdots \cup V_k$. Then, we construct an equivalent instance $G' = (V', E', s, t)$ of Shortest Secluded Path on edge-weighted graphs. We first define $V' = V \cup \{s, t\} \cup V_E \cup W$ where $V_E = \{v_e \mid e \in E\}$ and $W = \{w_h \mid h \in [k-1]\}$. Then we connect s to all vertices in V_1 and t to all vertices in V_k with edges of weight 1. Moreover, for each $h \in [k-1]$, we connect w_h to all vertices in $V_h \cup V_{h+1}$ with edges of weight 1. Finally, for $e = \{u, v\} \in E$, we connect v_e to u and v with edges of weight $k+1$. Figure 4 illustrates the constructed graph G'.

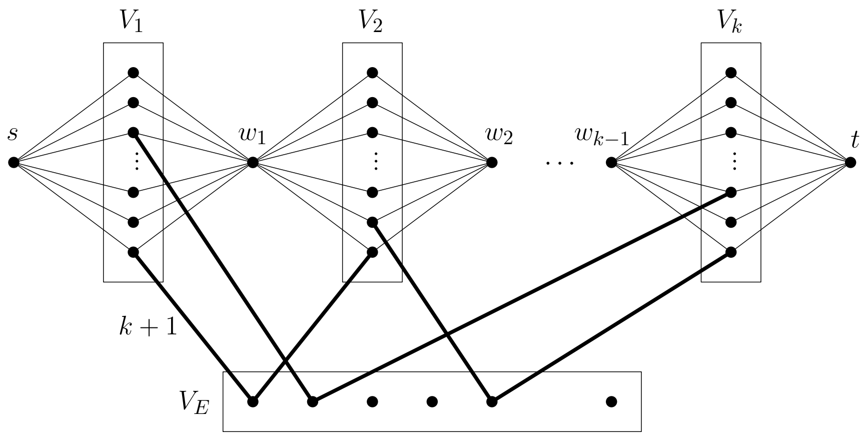

Figure 4 The reduction from Multicolored Clique to Shortest Secluded Path in Theorem 19. The weight of each thin edge is 1 and the weight of each bold edge is $k+1$.

Clearly, G' can be constructed in polynomial time. By the construction of G', the shortest path distance between s and t is $2k$. In the following, we show that there is a clique of size k in G if and only there is a shortest s-t path P of weight $2k$ satisfying $|N(V'(P))| \le rk - \binom{k}{2} + n - k$ in G' where $V'(P)$ is the set of vertices of P on G'.

Let $C = \{v_1, v_2, \ldots, v_k\}$ be a clique of size k in G, where $v_i \in V_i$. We define an s-t path $P = \langle s, v_1, w_1, v_2, w_2, \ldots, w_{k-1}, v_k, t\rangle$ of weight $2k$. Since all edges in $G'[V' \setminus V_E]$ have weights 1, the weight of P is $2k$, and thus P is a shortest s-t path in G'.

We next show P satisfies $|N(V'(P))| \le rk - \binom{k}{2} + n - k$. We observe that $V \setminus V'(P) \subseteq N(V'(P))$ holds. Since $|V'(P) \cap V| = k$, the neighborhood $N(V'(P))$ of P includes $n - k$ vertices in $V \setminus V'(P)$. Furthermore, since G is an r-regular graph, any vertex in V has r neighbors in V_E in G'. Since C is a clique in G, for $1 \le i < j \le k$, any pair $v_i, v_j \in C$ has an edge $\{v_i, v_j\}$. Thus, v_i and v_j have a common neighbor $v_{\{v_i, v_j\}} \in V_E$. Therefore, P has only $rk - \binom{k}{2}$ neighbors in V_E. Consequently, $|N(V'(P))| = rk - \binom{k}{2} + n - k$ holds.

Conversely, suppose that there is a shortest s-t path P of weight $2k$ satisfying $|N(V'(P))| \le rk - \binom{k}{2} + n - k$ in G'. If P contains any vertex in V_E, the weight of P is at least $2k + 2 > 2k$. Thus, P does not contain any vertex in V_E. Hence, by the construction of G', P must pass through $w_1, \cdots, w_{k-1}$ and exactly one vertex in each $V_1, \cdots, V_k$, and $V \setminus V(P) \subseteq N(V'(P))$ holds. Consequently, $|N(V(P)) \setminus V_E| = n - k$ and P has at most $rk - \binom{k}{2}$ neighbors in V_E. Since every vertex in $V'(P) \cap V$ has r neighbors in V_E, they must share $\binom{k}{2}$ vertices in V_E. Since $|V'(P) \cap V| = k$, every pair of $u, v \in V'(P) \cap V$ must share $v_{\{u,v\}} \in V_E$, which implies that there exists edge $\{u, v\}$ in G. This means that $V'(P) \cap V$ forms a clique of size k, which completes the proof. ◀

Finally, we present an XP algorithm for Shortest Secluded Path that runs in $n^{O(d)}$ time where d is the shortest path distance between s and t in the input graph.

▶ **Theorem 20.** *Shortest Secluded Path on edge-weighted graphs can be solved in* $n^{O(d)}$ *time, where* d *is the shortest path distance between* s *and* t *in the input graph.*

Proof. Given an input graph $G = (V, E)$, we first compute the shortest path distance $d = \mathrm{dist}(s, t)$ between s and t by Dijkstra's algorithm [7]. Since the edge weights are nonnegative integers, any shortest s-t path contains at most d edges. Thus, all shortest s-t paths can be enumerated in $n^{O(d)}$ time. Since the neighborhood of a shortest s-t path can be computed in polynomial time, Shortest Secluded Path can be solved in $n^{O(d)}$ time. ◀

7 Conclusion

In this paper, we investigated the structural parameterizations of Short Secluded Path and the computational complexity of Shortest Secluded Path.

For the structural parameterizations of Short Secluded Path, we presented an XP algorithm parameterized by cliquewidth and FPT algorithms parameterized by neighborhood diversity and twin cover number, respectively. Regarding Shortest Secluded Path, we first proposed a polynomial-time algorithm for unweighted graphs. For edge-weighted graphs, however, we proved that the problem is W[1]-hard but is in XP when parameterized by the shortest path distance between s and t.

A natural future direction is to investigate the parameterized complexity of Short Secluded Path with respect to other structural parameters, such as the cluster vertex deletion number and modular-width. Furthermore, improving the running times of our algorithms is an interesting challenge. In particular, given an r-expression tree, our XP algorithm parameterized by cliquewidth runs in $n^{O(r^2)}$ time. It is worth considering whether this running time can be improved to $n^{O(r)}$. It would also be interesting to design a single-exponential FPT algorithm parameterized by the twin cover number, since our current algorithm runs in $2^{O(\mathrm{tc}(G)^2)} n^{O(1)}$ time.

References

1 R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

2 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79(3):708–741, 2017. doi:10.1007/S00453-016-0222-Z.

3 Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Math. Program.*, 73(2):129–174, May 1996. doi:10.1007/BF02592101.

4 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Alexsander Andrade de Melo, Celina M. H. de Figueiredo, and Uéverton S. Souza. On the computational difficulty of the terminal connection problem. *RAIRO Theor. Informatics Appl.*, 57:3, 2023. doi:10.1051/ITA/2023002.

7 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.

8 Huib Donkers, Bart M. P. Jansen, and Jari J. H. de Kroon. Finding k-secluded trees faster. *J. Comput. Syst. Sci.*, 138:103461, 2023. doi:10.1016/J.JCSS.2023.05.006.

9 David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

10 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

11 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory Comput. Syst.*, 61(3):795–819, 2017. doi:10.1007/S00224-016-9717-X.

12 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

13 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

14 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. doi:10.46298/DMTCS.2136.

15 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. *J. Comput. Syst. Sci.*, 113:101–124, 2020. doi:10.1016/J.JCSS.2020.05.006.

16 Tesshu Hanaka and Yasuaki Kobayashi. Finding a minimum spanning tree with a small non-terminal set. *Theor. Comput. Sci.*, 1033:115092, 2025. doi:10.1016/J.TCS.2025.115092.

17 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

18 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

19 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.

20 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

21 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length- and neighborhood-constrained path problems. *Inf. Process. Lett.*, 156:105913, 2020. doi:10.1016/J.IPL.2019.105913.

22 Nadym Mallek, Jonas Schmidt, and Shaily Verma. A parameterized study of secluded structures in directed graphs. *CoRR*, abs/2502.06048, 2025. doi:10.48550/arXiv.2502.06048.

23 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

24 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

25 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory B*, 96(4):514–528, 2006. doi:10.1016/J.JCTB.2005.10.006.

26 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

27 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discret. Optim.*, 30:20–50, 2018. doi:10.1016/J.DISOPT.2018.05.002.

28 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. Parameterized algorithms and data reduction for the short secluded s-t-path problem. *Networks*, 75(1):34–63, 2020. doi:10.1002/NET.21904.