A Parameterized Study of Secluded Structures in Directed Graphs

Jonas Schmidt
Bocconi University, Milan, Italy

Shaily Verma
Hasso Plattner Institute, Potsdam, Germany

Nadym Mallek
Hasso Plattner Institute, Potsdam, Germany

Abstract

Given an undirected graph G and an integer k, the Secluded Π-Subgraph problem asks you to find a maximum size induced subgraph that satisfies a property Π and has at most k neighbors in the rest of the graph. This problem has been extensively studied; however, there is no prior study of the problem in directed graphs. This question has been mentioned by Jansen et al. [ISAAC'23].

In this paper, we initiate the study of Secluded Subgraph problems in directed graphs by incorporating different notions of neighborhoods: in-neighborhood, out-neighborhood, and their union. Formally, we call these problems {In, Out, Total}-Secluded Π-Subgraph, where given a directed graph G and an integer k, we want to find an induced subgraph satisfying Π of maximum size that has at most k in/out/total-neighbors in the rest of the graph, respectively. We investigate the parameterized complexity of these problems for different properties Π. In particular, we prove the following parameterized results:

- We design an FPT algorithm for the Total-Secluded Strongly Connected Subgraph problem when parameterized by k.
- We show that the Out-Secluded $\mathcal{F}$-Free Subgraph problem with parameter k is W[1]-hard, where $\mathcal{F}$ is a family of directed graphs except any subgraph of a star graph whose edges are directed towards the center. This result also implies that In/Out-Secluded DAG is W[1]-hard, unlike the undirected variants of the two problems, which are FPT.
- We design an FPT-algorithm for In/Out/Total-Secluded α-Bounded Subgraph when parameterized by k, where α-bounded graphs are a superclass of tournaments.
- For undirected graphs, we improve the best-known FPT algorithm for Secluded Clique by providing a faster FPT algorithm that runs in time $1.6181^k n^{\mathcal{O}(1)}$.

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

Keywords and phrases Secluded Subgraph, Parametrized Complexity, Directed Graphs, Strong Connectivity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.53

Related Version *Full Version*: https://arxiv.org/abs/2502.06048 [21]

1 Introduction

Finding substructures in graphs that satisfy specific properties is a ubiquitous problem. This general class of problems covers many classical graph problems such as finding maximum cliques, Steiner trees, or even shortest paths. Another compelling property to look for in a substructure is its isolation from the remaining graph. This motivated Chechik et al. [1], to introduce the concept of *secluded subgraphs*. Formally, in the Secluded Π-Subgraph problem, given an undirected graph G, the goal is to find a maximum size subset of vertices $S \subseteq \mathrm{V}(G)$ such that the subgraph induced on S fulfills a property Π and has a neighborhood $|\mathrm{N}(S)| \leq k$ where k is a natural number.

Table 1 Our main results for directed and undirected problems. WC stands for weakly connected. All FPT results are with respect to parameter k and all hardness results are with respect to parameter $k + w$. For the entry marked with $*$, the undirected algorithm from [15] immediately generalizes to the total-secluded setting (for finite $\mathcal{F}$). The problems marked with (?) are open.

Property Π	**In- / Out-Secluded**		**Total-Secluded**	
WC $\mathcal{F}$-Free Subgraph	W[1]-hard	Thm. 2	FPT*	[15]
WC DAG	W[1]-hard	Cor. 3	?	
α-Bounded Subgraph	FPT	Thm. 4	FPT	Thm. 5
Strongly Connected Subgraph	?		FPT	Thm. 1
Clique	FPT in time $1.6181^k n^{\mathcal{O}(1)}$			Thm. 6

This problem has been studied extensively for various properties Π such as paths [23, 17, 8, 1], Steiner trees [1, 8], induced trees [7, 11], subgraphs free of forbidden induced subgraphs [11, 15], and more [22]. Most of these studies focus on the parameterized setting, due to the strong relation to vertex deletion and separator problems, which are foundational in parameterized complexity. Our problem fits in that category since the neighborhood can be considered a $(S, V \setminus S)$-separator.

While the undirected Secluded Π-Subgraph problem has been explored and widely understood in prior work [15, 11, 7], the directed variant has not yet been studied, although it is a natural generalization and was mentioned as an interesting direction by Jansen et al. [15]. Directed graphs naturally model real-world systems with asymmetric interactions, such as social networks with unidirectional follow mechanisms or information flow in communication systems. Furthermore, problems such as Directed Feedback Vertex Set, Directed Multicut, and Directed Multiway Cut underline how directedness can make a fascinating and insightful difference when it comes to parameterized complexity [2, 13, 19, 4]. These problems and their results provide a ground for studying directed secluded subgraph problems, motivating the need to investigate them systematically.

In this paper, we introduce three natural directed variants of the Secluded Π-Subgraph problem, namely Out-Secluded Π-Subgraph, In-Secluded Π-Subgraph, and Total-Secluded Π-Subgraph. These problems limit either the out-neighborhood of S, its in-neighborhood, or the union of both. The out/in-neighborhood of a set S is the set of vertices in $\mathrm{V}(G) \setminus S$ reachable from S via an outgoing/incoming edge. The problems corresponding to different types of neighborhoods can be encountered in real-life networks. In privacy-aware social network analysis, one might aim to identify a community with minimal external exposure. Similarly, robust substructures with limited connectivity to vulnerable components are critical in cybersecurity. These real-world motivations further emphasize the need to explore and formalize directed variants of the Secluded Π-Subgraph problem.

For any property Π, we formulate our general problem as follows. Notice that in-secluded and out-secluded are equivalent for all properties that are invariant under the transposition of the edges. For this reason, we mostly focus on total-neighborhood and out-neighborhood.

X-Secluded Π-Subgraph $(\mathrm{X} \in \{\mathrm{In}, \mathrm{Out}, \mathrm{Total}\})$

Parameter: An integer $k \in \mathbb{N}$
Input: A directed graph G with vertex weights $\omega \colon V \to \mathbb{N}$ and an integer $w \in \mathbb{N}$
Output: An k-X-secluded set $S \subseteq \mathrm{V}(G)$ of weight $\omega(S) \geq w$ that satisfies Π, or report that none exists.

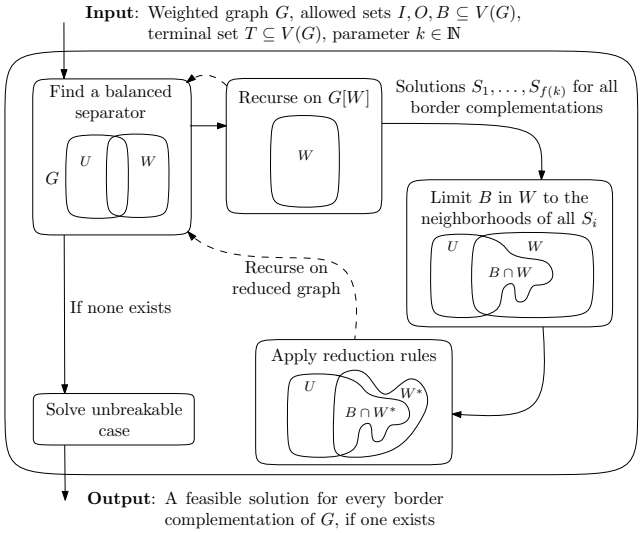


Figure 1 An illustration of the general recursive understanding algorithm used in Section 2. There are two recursive calls in total, highlighted with dashed arrows. As defined later, only vertices inside B are allowed to be in the neighborhood of a solution. W is chosen to be the side of the separation with a smaller intersection with T.

Our Contribution

In this subsection, we present the results we obtained for the problem. See Table 1 for an overview of the results we obtained in this paper.

Strongly Connected Subgraph. We show that the Total-Secluded Strongly Connected Subgraph problem is fixed-parameter tractable when parameterized with k. Precisely, we prove the following result:

▶ **Theorem 1.** *Total-Secluded Strongly Connected Subgraph is solvable in time* $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$.

We design our FPT algorithm for Total-Secluded Strongly Connected Subgraph problem using recursive understanding, a technique introduced by [12] and recently used successfully for various parameterized problems [3, 11, 6, 16]. Specifically, [16] prove a meta-result stating that if a problem is FPT on highly-connected graphs and expressible in *Counting Monadic Second-Order Logic*, it is also FPT on general graphs. Thereby, this theorem allows to shortcut the analysis and algorithm for the breakable case of recursive understanding. However, it comes at the price of being nonconstructive and not giving a concrete bound on the runtime. For this reason, it is still valuable to apply recursive understanding directly. In future work, it could be promising to apply and generalize this theorem for directed graphs. We visualize the overall structure of the algorithm in Figure 1.

On a high level, recursive understanding algorithms work by first finding a small balanced separator of the underlying undirected graph. If no suitable balanced separator exists, the graph must be highly connected, which makes the problem simpler to solve. In the other case, we reduce and simplify one side of the separator while making sure to keep an equivalent set of solutions in the whole graph. By choosing parameters in the right way, this process reduces one side of the separator enough to invalidate the balance of the separator. Therefore, we have made progress and can iterate with another balanced separator or reach the base case.

In our case, looking for a separator of size at most k makes the framework applicable. Crucially, this is because in any secluded subgraph $G[S]$, where $S \subseteq \mathrm{V}(G)$, the neighborhood $\mathrm{N}(S)$ acts as a separator between S and $\mathrm{V}(G) \setminus \mathrm{N}[S]$. Therefore, if no balanced separator of size at most k exists, we can deduce that either S itself or $\mathrm{V}(G) \setminus S$ must have a small size. This observation makes the problem significantly easier to solve in this case, using the color coding technique developed in [3].

In the other case, we can separate our graph into two balanced parts, U and W, with a separator P of size $|P| \leq k$. Now, our goal is to solve the same problem recursively for one of the sides, say W and replace that side with an equivalent graph of bounded size that only contains all necessary solutions. However, finding subsets of solutions is not the same as finding solutions; the solution S for the whole graph could heavily depend on including some vertices in U. That being said, the different options for this influence are limited. At most 2^k different subsets X of P could be part of the solution. For any such X, the solution can only interact across P in a limited number of ways. For finding strongly connected subgraphs, we have to consider for which pair $(x_1, x_2) \in X \times X$ there already is a x_1-x_2-path in the U-part of the complete solution S. This allows us to construct a new instance for every such possibility by encoding X and the existing paths into W. These instances are called *boundary complementations*. We visualize the idea of this construction in Figure 2.

Fundamentally, we prove that an optimal solution for the original graph exists, that coincides in W with an optimal solution to the boundary complementation graph in which U is replaced. Hence, we restrict the space of solutions to only those whose neighborhood in W coincides with the neighborhood of an optimal solution to some boundary complementation. The restricted instance consists of a bounded-size set B of vertices that could be part of $\mathrm{N}(S)$ and components in $W \setminus B$ that can only be included in S completely or not at all. We introduce graph *extensions* to formalize when exactly these components play the same role in a strongly connected subgraph. Equivalent extensions can then be merged and compressed into equivalent extensions of bounded size. In total, this guarantees that W is shrunk enough to invalidate the previous balanced separator, and we can restart the process.

$\mathcal{F}$-Free Subgraph. For any family of graphs $\mathcal{F}$, a graph G is called $\mathcal{F}$-Free if it does not contain any graph in $\mathcal{F}$ as an induced subgraph. Depending on the context both $\mathcal{F}$ and G are either both undirected or both directed. The widely-studied Secluded $\mathcal{F}$-Free Subgraph problem on undirected graphs is FPT (with parameter k) using recursive understanding [11] or branching on important separators [15] when we restrict it to connected solutions. We study the directed version of the problem and surprisingly, it turns out to be W[1]-hard for almost all forbidden graph families $\mathcal{F}$ even with respect to the parameter $k + w$. Precisely, we prove the following theorem.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a non-empty set of directed graphs such that no $F \in \mathcal{F}$ is a subgraph of an inward star. Then, Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph is W[1]-hard with respect to the parameter $k + w$ for unit weights.*

We establish an almost complete dichotomy that highlights the few cases of families $\mathcal{F}$ for which the problem remains tractable. One of these exceptions is if $\mathcal{F}$ contains an edgeless graph of any size, where we employ our algorithm for Out-Secluded α-Bounded Subgraph. Theorem 2 also implies the following result for the directed variant of Secluded Tree problem.

▶ **Corollary 3.** *Out-Secluded Weakly Connected DAG is W[1]-hard with parameter $k + w$ for unit weights.*

α-Bounded Subgraph & Clique. In the undirected setting, the Secluded Clique problem is natural and has been studied specifically. There is an FPT-algorithm running in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ through contracting twins [11]. The previous best algorithm however uses the general result for finding secluded $\mathcal{F}$-free subgraphs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ [15]. By using important separators, they require time at least $4^k n^{\mathcal{O}(1)}$. This property is naturally generalizable to directed graphs via tournament graphs. We go one step further.

The independence number of an undirected or directed graph G is the size of the maximum independent set in G (or its underlying undirected graph). If G has independence number at most α, we also call it *α-bounded*. This concept has been used to leverage parameterized results from the simpler tournament graphs to the larger graph class of α-bounded graphs [20, 10, 18]. We prove the following results:

▶ **Theorem 4.** *Out-Secluded α-Bounded Subgraph is solvable in time $(2\alpha+2)^k n^{\alpha+\mathcal{O}(1)}$.*

▶ **Theorem 5.** *Total-Secluded α-Bounded Subgraph is solvable in time $(\alpha+1)^k n^{\alpha+\mathcal{O}(1)}$.*

We achieve the goal via a branching algorithm, solving Secluded α-Bounded Subgraph for all neighborhood definitions in FPT time (Theorems 4 and 5). Our algorithm initially picks a vertex subset $U \subseteq \mathrm{V}(G)$ and looks only for solutions in the two-hop neighborhood of U. A structural property of α-bounded graphs guarantees that any optimal solution is found in this way. On a high level, the remaining algorithm depends on two branching strategies. First, we branch on forbidden structures in the two-hop neighborhood of U, to ensure that it becomes α-bounded. Second, we branch on farther away vertices to reach a secluded set.

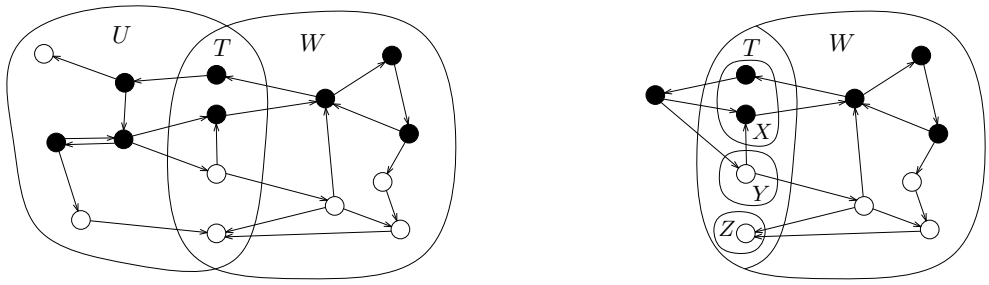
Note that Theorem 5 is inherently also an undirected result. Furthermore, the ideas behind the algorithm can in turn be used for the simpler undirected Secluded Clique problem. By a closer analysis of these two high-level rules, we arrive at a branching vector of $(1, 2)$ for Secluded Clique. This results in the following runtime, a drastic improvement on the previous barrier of $4^k n^{\mathcal{O}(1)}$.

▶ **Theorem 6.** *Secluded Clique is solvable in time $1.6181^k n^{\mathcal{O}(1)}$.*

Organization. We consider Total-Secluded Strongly Connected Subgraph and prove Theorem 1 in Section 2. The hardness result about Out-Secluded $\mathcal{F}$ Weakly Connected Subgraph in Theorem 2 is proved in Section 3. In Section 4, we give the algorithm for Out-Secluded α-Bounded Subgraph and prove Theorem 4. Due to space constraints, the algorithms and proofs of Theorems 5 and 6 can be found in the full version [21]. Proofs of statements marked with $(\star)$ are also deferred to the full version.

Notation. Let G be a directed graph. For a vertex $v \in \mathrm{V}(G)$, we denote the *out-neighborhood* by $\mathrm{N}^+(v) = \{u \mid (v,u) \in \mathrm{E}(G)\}$ and the *in-neighborhood* by $\mathrm{N}^-(v) = \{u \mid (u,v) \in \mathrm{E}(G)\}$. The *total-neighborhood* is defined as $\mathrm{N}(v) = \mathrm{N}^+(v) \cup \mathrm{N}^-(v)$. We use the same notation for sets of vertices $S \subseteq \mathrm{V}(G)$ as $\mathrm{N}^+(S) = \bigcup_{v \in S} \mathrm{N}^+(v) \setminus S$. Furthermore, for all definitions, we also consider their *closed* version that includes the vertex or vertex set itself, denoted by $\mathrm{N}^+[v] = \mathrm{N}^+(v) \cup \{v\}$. For a vertex set $S \subseteq \mathrm{V}(G)$, we write $G[S]$ for the subgraph induced by S or $G - S$ for the subgraph induced by $\mathrm{V}(G) \setminus S$. We also use $G - v$ instead of $G - \{v\}$.

When we refer to a component of a directed graph, we mean a component of the underlying undirected graph, that is, a maximal set that induces a weakly connected subgraph. In contrast, a strongly connected component refers to a maximal set that induces a strongly connected subgraph. For standard parameterized definitions, we refer to [5].



(a) A strongly connected subgraph S of a graph G with $k = 5$ neighbors. Black vertices are part of S.

(b) The boundary complementation that admits an equivalent feasible solution when setting $k' := k-2$.

Figure 2 A visualization of a solution in the original graph and a solution in a boundary complementation. Every partial solution in U can be represented by a boundary complementation.

2 Total-Secluded Strongly Connected Subgraph

In this section, we investigate the Total-Secluded Strongly Connected Subgraph problem, or TSSCS for short. First, we prove that the problem is NP-hard in general graphs, motivating analysis of its parameterized complexity.

▶ **Theorem 7** ($\star$). *TSSCS is NP-hard, even for unit weights.*

The proof of Theorem 7 also shows that TSSCS is W[1]-hard when parameterized by w, since w in the proof also only depends on the parameter for Clique. In the following subsections, we describe the recursive understanding algorithm to solve TSSCS parameterized by k. We follow the framework by [3, 11] and first introduce generalized problems in Section 2.1. In Section 2.2, we solve the case of unbreakable graphs. We introduce graph *extensions* in Section 2.3 as a framework to formulate our reduction rules and full algorithm in Section 2.4.

2.1 Boundaries and boundary complementations

In this subsection, we first define an additional optimization problem that is useful for recursion. Then, we describe a problem-specific *boundary complementation*. Finally, we define the auxiliary problem that our algorithm solves, which includes solving many similar instances from the optimization problem.

Max TSSCS

Input: A directed graph G, subsets $I, O, B \subseteq \mathrm{V}(G)$, a weight function $\omega\colon \mathrm{V}(G) \to \mathbb{N}$, and an integer $k \in \mathbb{N}$
Output: A set $S \subseteq \mathrm{V}(G)$ that maximizes $\omega(S)$ subject to $I \subseteq S$, $O \cap S = \emptyset$, $N(S) \subseteq B$, $|\mathrm{N}(S)| \leq k$, and $G[S]$ strongly connected, or report that no feasible solution exists.

Note that this problem generalizes the optimization variant of TSSCS by setting $I := O := \emptyset$ and $B := \mathrm{V}(G)$. However, Max TSSCS allows us to put more constraints on recursive calls, enforcing vertices to be included or excluded from the solution and neighborhood.

▶ **Definition 8** (Boundary Complementation). *Let $\mathcal{I} = (G, I, O, B, \omega, k)$ be a* Max TSSCS *instance. Let $T \subseteq \mathrm{V}(G)$ be a set of* boundary terminals *with a partition $X, Y, Z \subseteq T$ and let $R \subseteq X \times X$ be a relation on X. Then, we call the instance $(G', I', O', B, \omega', k')$ a* boundary complementation *of $\mathcal{I}$ and T if*

1. *G' is obtained from G by adding vertices $u_{(a,b)}$ for every $(a, b) \in R$ and edges $(a, u_{(a,b)})$, $(u_{(a,b)}, b)$, and for every $y \in Y$ additionally $(u_{(a,b)}, y)$,*
2. *$I' \coloneqq I \cup X \cup \{u_{(a,b)} \mid (a, b) \in R\}$,*
3. *$O' \coloneqq O \cup Y \cup Z$,*
4. *$\omega'(v) \coloneqq \omega(v)$ for $v \in \mathrm{V}(G)$ and $\omega'(u_{(a,b)}) \coloneqq 0$ for $(a, b) \in R$, and*
5. *$k' \leq k$.*

See Figure 2 for an example boundary complementation. The intuition here should be that if we take the union of G with any other graph H and only connect H to G at the vertices in T, then (X, Y, Z, R) encodes all possibilities of how a solution in $G \cup H$ could behave from G's point of view. So, for any solution S to Max TSSCS in $G \cup H$, there is some boundary complementation for G in which we can solve and exchange $S \cap G$ for that solution. Later, we prove a statement that is similar to this intuition.

To employ recursive understanding, we need a boundaried version of the problem. Intuitively, this problem is the same as the previous Max TSSCS but for a small part of the graph we want to try out every possibility, giving many very similar instances. This small part will later represent a separator to a different part of the graph.

Boundaried Max TSSCS

Input: A Max TSSCS instance $\mathcal{I} = (G, I, O, B, \omega, k)$ and a set of boundary terminals $T \subseteq \mathrm{V}(G)$ with $|T| \leq 2k$
Output: A solution to Max TSSCS for each boundary complementation $\mathcal{I}'$ of $\mathcal{I}$ and T, or report that no solution exists.

To even have a chance to solve this problem, we need to make sure that there are not too many boundary complementations. The following lemma bounds that number in terms of k.

▶ **Lemma 9** ($\star$). *For a Max TSSCS instance (G, I, O, B, ω, k) and $T \subseteq \mathrm{V}(G)$, there are at most $3^{|T|}2^{|T|^2}(k+1)$ many boundary complementations, which can be enumerated in time $2^{\mathcal{O}(|T|^2)}n^{\mathcal{O}(1)}$.*

2.2 Unbreakable Case

This subsection gives the algorithm for the base case of our final recursive algorithm, when no balanced separator exists. We start by giving the definitions of separations and unbreakability.

▶ **Definition 10** (Separation). *Given two sets $A, B \subseteq \mathrm{V}(G)$ with $A \cup B = \mathrm{V}(G)$, we say that (A, B) is a* separation of order $|A \cap B|$ *if there is no edge with one endpoint in $A \setminus B$ and the other endpoint in $B \setminus A$.*

▶ **Definition 11** (Unbreakability). *Let $q, k \in \mathbb{N}$. An undirected graph G is* (q, k)-unbreakable *if for every separation (A, B) of G of order at most k, we have $|A \setminus B| \leq q$ or $|B \setminus A| \leq q$.*

Due to space constraints, we defer any further details on the algorithm for the unbreakable case to the full version [21]. It uses standard color coding techniques from [3, 11].

▶ **Theorem 12** ($\star$). *Boundaried Max TSSCS on (q, k)-unbreakable graphs can be solved in time $2^{\mathcal{O}(k^2 \log(q))}n^{\mathcal{O}(1)}$.*

2.3 Compressing Graph Extensions

Before we give the complete algorithm, we define a routine that compresses a part of the graph. We aim for two crucial properties in the compressed part. First, the part after compression should be equivalent to the part before compression in terms of which strongly connected components can be formed. Second, we want the size of the compressed part to be functionally bounded by the size of remaining graph.

To achieve this goal, we first formally define sufficient properties to reason about this equivalence and bound the number of equivalence classes. First, we define the notion of a graph *extension*, a way to extend one graph with another. The extension will play the role of the compressed part of the graph. This concept allows us to speak more directly about graph properties before and after exchanging a part of the graph with a different one.

▶ **Definition 13** (Extension). *Given a directed graph G, we call a pair (D, E_{GD}) an* extension *of G if D is a directed graph and $E_{GD} \subseteq (\mathrm{V}(G) \times \mathrm{V}(D)) \cup (\mathrm{V}(D) \times \mathrm{V}(G))$ is a set of pairs between G and D. We name the graph $Ext_G(D, E_{GD}) \coloneqq (\mathrm{V}(G) \cup \mathrm{V}(D), \mathrm{E}(G) \cup \mathrm{E}(D) \cup E_{GD})$, that can be created from the extension, G* extended by *(D, E_{GD}).*

We use extensions to construct extended graphs. Intuitively, an extension of G is a second graph D together with an instruction E_{GD} on how to connect D to G.

Next, we identify three important attributes of extensions in our context. Later, we show that these give a sufficient condition on when two extensions form the same strongly connected subgraphs. For this, consider a directed graph G with an extension (D, E_{GD}). For $U \subseteq \mathrm{V}(D)$, we write $\mathrm{N}^-_{E_{GD}}(U)$ as a shorthand for $\mathrm{N}^-_{\mathrm{Ext}_G(D,E_{GD})}(U)$, that is, all $v \in \mathrm{V}(G)$ with $(v, u) \in E_{GD}$ for some $u \in U$. Define $\mathrm{N}^+_{E_{GD}}(v)$ analogously. Write $\mathrm{SCC}(D)$ for the *condensation* of D, where every strongly connected component C of D is contracted into a single vertex. Define $\mathcal{S}(D, E_{GD}), \mathcal{T}(D, E_{GD}) \subseteq 2^{\mathrm{V}(G)}$ such that

$$\mathcal{S}(D, E_{GD}) \coloneqq \left\{ \mathrm{N}^-_{E_{GD}}(U) \;\middle|\; U \subseteq \mathrm{V}(D) \text{ is a source component in } \mathrm{SCC}(D) \right\} \text{ and}$$
$$\mathcal{T}(D, E_{GD}) \coloneqq \left\{ \mathrm{N}^+_{E_{GD}}(U) \;\middle|\; U \subseteq \mathrm{V}(D) \text{ is a sink component in } \mathrm{SCC}(D) \right\},$$

that is, for every strongly connected source component C in $\mathrm{SCC}(D)$, $\mathcal{S}(D, E_{GD})$ contains the set of all $v \in \mathrm{V}(G)$ such that $(v, u) \in E_{GD}$ for some $u \in C$ and analogously for $\mathcal{T}(D, E_{GD})$. Furthermore, define

$$\mathcal{C}(D, E_{GD}) \coloneqq \left\{ (a, b) \in \mathrm{V}(G)^2 \;\middle|\; \text{there is a } d_1\text{-}d_2\text{-path in } D \text{ with } (a, d_1), (d_2, b) \in E_{GD} \right\},$$

that is, all (a, b) such that there is an a-b-path in $\mathrm{Ext}_G(D, E_{GD})$, whose intermediate vertices and edges belongs to D. Refer to Figure 3 for examples of extensions and the three sets.

▶ **Definition 14** (Equivalent Extensions). *Let G be a directed graph. We say that two extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}) of G are* equivalent *if*

$$(\mathcal{S}(D_1, E_{GD_1}), \mathcal{T}(D_1, E_{GD_1}), \mathcal{C}(D_1, E_{GD_1})) = (\mathcal{S}(D_2, E_{GD_2}), \mathcal{T}(D_2, E_{GD_2}), \mathcal{C}(D_2, E_{GD_2})).$$

Clearly, extension equivalence defines an equivalence relation. The next statement reveals the motivation behind the definition of extension equivalence. It gives us a sufficient condition for two extensions being exchangeable in a strongly connected subgraph.

▶ **Lemma 15.** *Let G be a directed graph with two equivalent extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}). Let $U \subseteq \mathrm{V}(G)$ be nonempty such that the extended graph $Ext_{G[U]}(D_1, E_{G[U]D_1})$ is strongly connected. Then $Ext_{G[U]}(D_2, E_{G[U]D_2})$ is also strongly connected.*

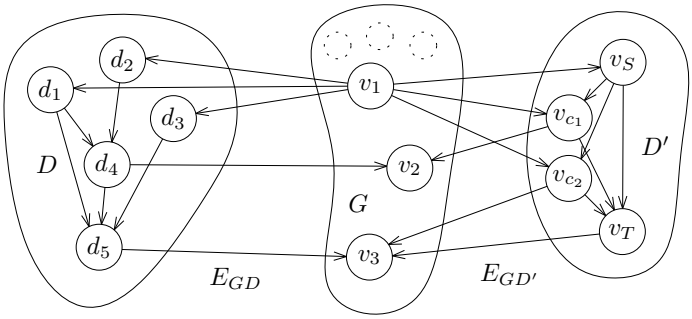


Figure 3 Two example extensions of a graph G. Observe that $\mathcal{S}(D, E_{GD}) = \{\{v_1\}\}$, $\mathcal{T}(D, E_{GD}) = \{\{v_3\}\}$, and $\mathcal{C}(D, E_{GD}) = \{(v_1, v_2), (v_1, v_3)\}$. The extension $(D', E_{GD'})$ not only has the same sets $\mathcal{S}, \mathcal{T}, \mathcal{C}$ and is thereby equivalent; it is also the compressed extension of (D, E_{GD}). Since all sources d_1, d_2, d_3 have the same in-neighborhood, they are represented by the single vertex v_S.

Proof. We construct a v_1-v_2-path for all $v_1, v_2 \in U \cup \mathrm{V}(D_2)$ that only uses edges in $\mathrm{E}(D_2)$, E_{GD_2}, and $G[U]$ by case distinction.

Paths $U \to U$. Let $u_1, u_2 \in U$. If there is a path from u_1 to u_2 in $G[U]$, this path also exists after exchanging (D_1, E_{GD_1}) to (D_2, E_{GD_2}). If the path passes through D_1, since $\mathcal{C}(D_1, E_{GD_1}) = \mathcal{C}(D_2, E_{GD_2})$, we can exchange all subpaths through D_1 by subpaths through D_2.

Paths $\mathrm{V}(D_2) \to U$. Let $v \in \mathrm{V}(D_2), u \in U$. We construct a v-u-path by first walking from v to any sink component T in $\mathrm{SCC}(D_2)$. If there is no edge $(t, u') \in E_{GD_2}$ with $t \in T, u' \in U$ that we can append, since $\mathcal{T}(D_1, (D_1, E_{GD_1})) = \mathcal{T}(D_2, (D_2, E_{GD_2}))$, there must also be a sink component in $\mathrm{SCC}(D_1)$ with no outgoing edge to U. However, this is a contradiction to the fact that $\mathrm{Ext}_{G[U]}(D_1, E_{G[U]D_1})$ is strongly connected with nonempty U. Therefore, we can find a (t, u') to append for some $t \in T, u' \in U$. From u', there is already a path to u, as proven in the first case.

Paths $U \to \mathrm{V}(D_2)$. Next, we construct a u-v-path backwards by walking from v backwards to a source s in D_2. Analogously, there is an edge $(u', s) \in E_{GD_2}$ for some $u' \in U$ since $\mathcal{S}(D_1, E_{GD_1}) = \mathcal{S}(D_2, E_{GD_2})$, which we append. From u, there is a path to u', as proven in the first case, which we prepend to the rest of the path.

Paths $\mathrm{V}(D_2) \to \mathrm{V}(D_2)$. Let $v_1, v_2 \in \mathrm{V}(D_2)$. To construct a v_1-v_2-path, we can just walk from v_1 to any $u \in U$ and from there to v_2 as shown before. ◀

Furthermore, observe that the union of two extensions creates another extension where source, sink and connection sets correspond exactly to the union of the previous sets. Hence, the union of two equivalent extensions will again be equivalent. This fact is formalized in the next observation and will turn out useful in later reduction rules.

▶ **Observation 16.** *Let G be a directed graph with two equivalent extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}). Consider the extension defined by $D \coloneqq (\mathrm{V}(D_1) \cup \mathrm{V}(D_2), \mathrm{E}(D_1) \cup \mathrm{E}(D_2))$ and $E_{GD} \coloneqq E_{GD_1} \cup E_{GD_2}$. Then (D, E_{GD}) is equivalent to (D_1, E_{GD_1}) and (D_2, E_{GD_2}).*

Now, we finally define our compression routine, which compresses an extension to a bounded size equivalent extension. If an extension is strongly connected, it is easy to convince yourself that it is always possible to compress the extension to a single vertex. Otherwise, we add one source vertex per neighborhood set in $\mathcal{S}(D, E_{GD})$ as well as one sink vertex per neighborhood set in $\mathcal{T}(D, E_{GD})$, realizing the same $\mathcal{S}$ and $\mathcal{T}$. Then, we add

vertices in between suitable source and sink vertices to realize exactly the same connections in $\mathcal{C}$ without creating additional ones. The result of a compression is visualized in Figure 3. Now, we describe the procedure formally.

Let G be a directed graph with an extension (D, E_{GD}). Our *compression routine* returns an extension that we call *compressed extension*, denoted as $\mathrm{Comp}_G(D, E_{GD})$.

Compression Routine.

- If D is strongly connected, we contract D to one vertex v and remove self-loops and multiple edges. We adjust E_{GD} by using v instead of $\mathrm{V}(D)$ and removing multiple edges.
- Otherwise, $\mathrm{Comp}_G(D, E_{GD}) \coloneqq (D', E_{GD'})$, where $(D', E_{GD'})$ is an extension such that

$$\begin{aligned}\mathrm{V}(D') &\coloneqq \{v_S \mid S \in \mathcal{S}(D, E_{GD})\} \cup \{v_T \mid T \in \mathcal{T}(D, E_{GD})\} \cup \{v_c \mid c \in \mathcal{C}(D, E_{GD})\},\\ E_{GD'} &\coloneqq \{(s, v_S) \mid S \in \mathcal{S}(D, E_{GD}), s \in S\} \cup \{(v_T, t) \mid T \in \mathcal{T}(D, E_{GD}), t \in T\}\\ &\quad \cup \{(a, v_c), (v_c, b) \mid c = (a, b) \in \mathcal{C}(D, E_{GD})\}.\end{aligned}$$

  To define $\mathrm{E}(D)$, consider every source component C_s and sink component C_t in $\mathrm{SCC}(D)$ such that C_t is reachable from C_s in D. Let $S \coloneqq \mathrm{N}^-_{E_{GD}}(C_s)$ and $T \coloneqq \mathrm{N}^-_{E_{GD}}(C_t)$ be the corresponding sets in $\mathcal{S}(D, E_{GD})$ and $\mathcal{T}(D, E_{GD})$.
  - Add the edge (v_S, v_T) to $\mathrm{E}(D)$.
  - For every $c = (a, b) \in \mathcal{C}(D, E_{GD})$ that satisfies $(s, b) \in \mathcal{C}(D, E_{GD})$ and $(a, t) \in \mathcal{C}(D, E_{GD})$ for every $s \in S, t \in T$, add the edges (v_S, v_c) and (v_c, v_T) to $\mathrm{E}(D)$.

Now we go on to prove the properties that are maintained while compressing. Then, we bound the size of a compressed extension and thus also the number of equivalence classes.

▶ **Lemma 17.** *Let G be a directed graph with an extension (D, E_{GD}) and let $(D', E_{GD'})$ be the compressed extension of (D, E_{GD}). Then, the following are true.*

1. *If D is weakly connected, then D' is also weakly connected.*
2. *D is strongly connected if and only if D' is strongly connected.*
3. *$(D', E_{GD'})$ is equivalent to (D, E_{GD}).*

Proof. For the first property, assume that D is weakly connected. We know by definition that every sink in D' is reached by at least one source. Consider a v_c with $c = (a, b) \in \mathcal{C}(D, E_{GD})$. To show that v_c is connected to some v_S and v_T, consider the path from d_1 to d_2 in D that realizes this connection. There must be a source component C_S and a sink component C_T in $\mathrm{SCC}(D)$ such that d_1 is reachable from C_S and C_T is reachable from d_2. Therefore, any vertex in C_S can also reach d_2 and d_1 can reach every vertex in C_T. By definition of compression, these two components ensure that v_c is connected.

It remains to show that any source is reachable by any other source in the underlying undirected graph. Let $v_S, v_{S'}$ be two sources in D' with corresponding source components C, C' in $\mathrm{SCC}(D)$. Since D is weakly connected, there is a path from C to C' in the underlying undirected graph. Whenever the undirected path uses an edge in a different direction than the one before, we extend the path to first keep using edges in the same direction until a source or sink component is reached and then go back to the switching point. This new path can directly be transferred to D', where we only keep the vertices corresponding to source and sink components. By definition, this is still a path in D' that connects v_S to $v_{S'}$, and D' is weakly connected

The second property is simple to verify, since strongly connected graphs are by definition compressed to single vertices. If D is not strongly connected, D' will have at least one source and one sink that are not the same.

Regarding the equivalence, we create one source for every $S \in \mathcal{S}(D, E_{GD})$ with the same set of incoming neighbors and create no other sources. Therefore, $\mathcal{S}(D', E_{GD'}) = \mathcal{S}(D, E_{GD})$ and $\mathcal{T}(D', E_{GD'}) = \mathcal{T}(D, E_{GD})$ follows analogously. For every connection $c \in \mathcal{C}(D, E_{GD})$, we create v_c in D' that realizes this connection. Therefore, we know that $\mathcal{C}(D', E_{GD'}) \supseteq \mathcal{C}(D, E_{GD})$. Since v_c is only reachable from sources and reaches only sinks that do not give new connections, we arrive at $\mathcal{C}(D', E_{GD'}) = \mathcal{C}(D, E_{GD})$. ◀

We also bound the number of possible different compression outputs as well as their size.

▶ **Lemma 18** (⋆). *For a directed graph G, there can be at most $2^{2 \cdot 2^{|\mathrm{V}(G)|} + |\mathrm{V}(G)|^2}$ different compressed extensions. Furthermore, every compressed extension has at most $2^{|\mathrm{V}(G)|+1} + |\mathrm{V}(G)|^2$ vertices.*

In the past section, we have defined graph extensions and an equivalence relation on them that captures the role they can play in forming strongly connected subgraphs. We have presented a way to compress an extension such that its size only depends on the size of G, while remaining equivalent. In the next section, we will use this theory to design reduction rules for Boundaried Max TSSCS. Crucially, we view components outside of the set B as extensions, which allows us to keep only one compressed extension of each equivalence class.

2.4 Solving Boundaried Max TSSCS

We start by giving some reduction rules for a Boundaried Max TSSCS instance $\mathcal{I} = (G, I, O, B, \omega, k, T)$. Additionally, we assume that $T \subseteq B$ to ensure that we do not change T when changing $G - B$. This condition will always be satisfied in our algorithm. A (⋆) after a reduction rule denotes that its proof of safeness can be found in the full version [21].

The first reduction rule extends the sets I and O to whole components of $G - B$. This is possible since no solution can include only part of a component without its neighborhood intersecting the component. Remember that components always refer to weakly connected components.

▶ **Reduction Rule 19** (⋆). *Let Q be a component of $G - B$. If $Q \cap O \neq \emptyset$, set $O = O \cup \mathrm{N}[Q]$. If $\mathrm{N}[Q] \cap I \neq \emptyset$, set $I = I \cup Q$. If both cases apply, the instance has no solution.*

If this reduction rule is no longer applicable, every component in $G - B$ is either completely in I, completely in O, or intersects with none of the two.

From this point, we will use extensions from the previous section for components of $G - B$. Namely, for a component Q of $G - B$, let E_{BQ} be all the edges with exactly one endpoint in B and one in Q. Then, $(G[Q], E_{BQ})$ defines an extension of $G - Q$. For simplicity, we also refer to this extension as (Q, E_{BQ}). Hence, we also use $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{C}$, as well as Comp_{G-Q} for these extensions.

The next reduction rule identifies a condition under which a component Q can never be part of a solution, namely, if Q includes strongly connected components with no in-neighbors or no out-neighbors, which is exactly the case if the empty set is in $\mathcal{S}(Q, E_{BQ})$ or $\mathcal{T}(Q, E_{BQ})$.

▶ **Reduction Rule 20** (⋆). *Let Q be a component of $G - B$ such that $G[Q]$ is not strongly connected. If $\emptyset \in \mathcal{S}(Q, E_{BQ}) \cup \mathcal{T}(Q, E_{BQ})$, include Q into O.*

Note that after Reduction Rules 19 and 20 have been applied exhaustively, every source in Q has incoming edges from B, and every sink in Q has outgoing edges to B. Finally, we have one more simple rule, which removes vertices $v \in O \setminus B$. It relies on the fact that by Reduction Rule 19, we also have $\mathrm{N}(v) \subseteq O$.

▶ **Reduction Rule 21.** *If Reduction Rule 19 is not applicable, remove $O \setminus B$ from G.*

The previous reduction rules were useful to remove trivial cases and extend I and O. From now on, we assume that the instance is exhaustively reduced by Reduction Rules 19–21. Therefore, any component of $G - B$ is either contained in I or does not intersect I and O.

The next two rules will be twin type reduction rules that allow us to bound the number of remaining components. If there are two components of $G - B$ that form equivalent extensions it is enough to keep one of them, since they fulfill the same role in forming a strongly connected subgraph. The reduction rules rely on Lemma 15 and Observation 16 to show that equivalent extensions can replace each other and can be added to any solution.

▶ **Reduction Rule 22.** *Let Q_1, Q_2 be components of $G - B$ such that both $G[Q_1]$ and $G[Q_2]$ are not strongly connected and (Q_1, E_{BQ_1}) and (Q_2, E_{BQ_2}) are equivalent. Delete Q_2 and increase the weight of some $q \in Q_1$ by $\omega(Q_2)$. If $Q_2 \cap I \neq \emptyset$, set $I = I \cup Q_1$.*

Proof of Safeness. By the previous reduction rules, components of $G - B$ can only be included as a whole or not at all. Notice that since $\mathcal{C}(Q_1, E_{BQ_1}) = \mathcal{C}(Q_2, E_{BQ_2})$ and Reduction Rule 20, we get $\mathrm{N}(Q_1) = \mathrm{N}(Q_2)$. Let S be a solution to the old instance. We differentiate some cases.

If $S \cap (Q_1 \cup Q_2) = \emptyset$, we know that also $\mathrm{N}(S) \cap (Q_1 \cup Q_2) = \emptyset$. Therefore, the neighborhood size and strong connectivity of S do not change in the new instance, and it is also a solution. If S includes only one of Q_1 and Q_2, assume without loss of generality $S \cap (Q_1 \cup Q_2) = Q_1$, since Q_1 is not strongly connected by itself, the solution must include vertices of B. Because $\mathrm{N}(Q_1) = \mathrm{N}(Q_2)$, the solution must also include Q_2, a contradiction. If S includes both Q_1 and Q_2, we claim that $S' \coloneqq S \setminus Q_2$ is a solution for the new instance. The neighborhood size and weight clearly remain unchanged. Strong connectivity follows by Observation 16 and Lemma 15. The last two cases also show that if a solution had to include at least one of Q_1 and Q_2, that is, $(Q_1 \cup Q_2) \cap I \neq \emptyset$, any solution for the reduced instance must also include Q_1. Therefore, the adaptation to I is correct.

Let S be a solution to the reduced instance. Again, if S does not include vertices from Q_1, then S will immediately be a solution to the old instance. If $Q_1 \subseteq S$, then $S' \coloneqq S \cup Q_2$ will be a solution to the old instance by Observation 16 and Lemma 15. ◀

For strongly connected components, the rule is different, since we have to acknowledge the fact that they can be a solution by themselves. For every such component we destroy strong connectivity of smaller-weight equivalent component, which can then be reduced by Reduction Rule 22.

▶ **Reduction Rule 23.** *Let Q_1, Q_2 be components of $G - B$ that are also strongly connected with (Q_1, E_{BQ_1}) and (Q_2, E_{BQ_2}) equivalent and $\omega(Q_1) \geq \omega(Q_2)$. Add a vertex q_2' with edges (q_2, q_2') and (q_2', v), for all $q_2 \in Q_2$ and $v \in \mathrm{N}(Q_2)$. Set $\omega(q_2') = 0$.*

Proof of Safeness. Let S be a solution of the old instance. If $S \cap (Q_1 \cup Q_2) = \emptyset$, then S is also a solution for the new instance. If S includes only one of Q_1 and Q_2, then we must have $S \in \{Q_1, Q_2\}$, so $S' \coloneqq Q_1$ is a solution for the new instance with $\omega(S') \geq \omega(S)$. If S includes both Q_1 and Q_2, adding q_2' to S obviously gives a solution of the same weight.

For a solution of the reduced instance S, we can simply remove q_2' if it is inside for a solution to the old instance. ◀

Using both Reduction Rules 22 and 23 exhaustively makes sure that there are at most two components per extension equivalence class left. The last rule compresses the remaining components to equivalent components of bounded size.

Algorithm 1 The recursive understanding algorithm for Boundaried Max TSSCS.

def `Solve`$(G,\ I,\ O,\ B,\ \omega,\ k,\ T)$**:**
  $q \leftarrow 2^{2^{2^{ck^2}}}$ for a suitable constant c
  if G *is* $((2q+1)q2^k, k)$*-unbreakable* **then**
    return solve the problem using Theorem 12
  else
    $(U, W) \leftarrow (q,k)$-separation of G with $|T \cap W \setminus U| \leq k$
    $(\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, k, \tilde{T}) \leftarrow$ restriction to W with $\tilde{T} = (T \cap W) \cup (U \cap W)$
    $\mathcal{R} \leftarrow$ `Solve`$(\tilde{G},\ \tilde{I},\ \tilde{O},\ \tilde{B},\ \tilde{\omega},\ k,\ \tilde{T})$
    $\mathcal{N} \leftarrow \tilde{T} \cup \bigcup_{R \in \mathcal{R}} \mathrm{N}(R) \cap W$
    $\hat{B} \leftarrow (B \cap U) \cup (B \cap \mathcal{N})$
    $(G^*, I^*, O^*, \hat{B}^*, \omega^*, k^*, T^*) \leftarrow$ reduce $(G, I, O, \hat{B}, \omega, k, T)$ with Lemma 25
    return `Solve`$(G^*,\ I^*,\ O^*,\ \hat{B}^*,\ \omega^*,\ k^*,\ T^*)$
  end

▶ **Reduction Rule 24.** *Let Q be a component of $G - B$ that is not equal to its compressed extension. Replace Q by its equivalent compressed extension and set the weight such that $\omega(Q') = \omega(Q)$. If $Q \cap I \neq \emptyset$, set $I = I \cup Q'$.*

Proof of Safeness. Since Q is not strongly connected, it can only be part of a solution that includes some vertices from $G - Q$. Thus, we can apply Lemmas 15 and 17, and the old instance and the new instance have exactly the same strongly connected subgraphs. Because of $\mathcal{C}(Q, E_{BQ}) = \mathcal{C}(Q', E_{BQ'})$ and Reduction Rule 20, we get $\mathrm{N}(Q) = \mathrm{N}(Q')$. Since $\omega(Q') = \omega(Q)$, the rule is safe. ◀

This finally allows us to bound the size of $G - B$. In the next lemma, we summarize the progress of our reduction rules and apply the bounds from the previous section. Note that we need the stronger bound using the neighborhood of the components instead of simply B.

▶ **Lemma 25** (⋆). *Let $\mathcal{Q}$ be a set of components of $G - B$ with total neighborhood size $h := \left|\bigcup_{Q \in \mathcal{Q}} \mathrm{N}(Q)\right|$. Then we can reduce the instance, or there are at most $2^{2^{h+1}+h^2}\left(2^{h+1} + h^2\right) = 2^{\mathcal{O}(2^h)}$ vertices in $\mathcal{Q}$ in total.*

Executing the reduction rules in the proposed order guarantees termination after $\mathcal{O}(n)$ applications. The total execution takes at most $n^{\mathcal{O}(1)}$ time.

▶ **Lemma 26** ([11, Lemma 3])**.** *Given an undirected graph G, there is an algorithm with runtime $2^{\mathcal{O}(\min\{q,k\}\log(q+k))}n^{\mathcal{O}(1)}$ that either finds a (q,k)-separation of G or correctly reports that G is $((2q+1)q2^k, k)$-unbreakable.*

Now, we have all that it takes to solve our intermediate problem. The main idea of the algorithm is to shrink B to a bounded size, by solving the problem recursively. Once B is bounded, we apply our reduction rules by viewing components of $G - B$ as extensions, removing redundant equivalent extensions and compressing them. Thereby, we also bound the size and number of components of $G - B$ in terms of $|B|$ using Lemma 25. By choosing suitable constants, we can show that this decreases the total size of G, which will make progress to finally reduce it to the unbreakable case.

▶ **Theorem 27** (⋆). *Boundaried Max TSSCS can be solved in time $2^{2^{2^{\mathcal{O}(k^2)}}}n^{\mathcal{O}(1)}$.*

Proof. Let $\mathcal{I} = (G, I, O, B, \omega, k, T)$ be our Boundaried Max TSSCS instance. See Algorithm 1 for a more compact description of the algorithm. A high level display of the approach can be found in Figure 1. Define $q = 2^{2^{2^{ck^2}}}$ for a constant c. We later show that a suitable c must exist.

First, we run the algorithm from Lemma 26 on the underlying undirected graph of G with q and k. If it is $((2q+1)q2^k, k)$-unbreakable, we solve the instance directly using the algorithm from Theorem 12.

Therefore, assume that we have a (q,k)-separation (U, W). Without loss of generality, since $|T| \leq 2k$ we can assume that $|T \cap W \setminus U| \leq k$. Thus, we can construct a new instance to solve *the easier side* of the separation. Take $\tilde{G} = G[W], \tilde{I} = I \cap W, \tilde{O} = O \cap W, \tilde{B} = B \cap W$, write $\tilde{\omega}$ for the restriction of ω to W, and set $\tilde{T} = (T \cap W) \cup (U \cap W)$. Since $|U \cap W| \leq k$, also $\left|\tilde{T}\right| \leq 2k$ holds and $\tilde{I} \coloneqq (\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, k, \tilde{T})$ is a valid instance, which we solve recursively.

Let $\mathcal{R}$ be the set of solutions found in the recursive call. For $R \in \mathcal{R}$, define $N_R = \mathrm{N}(R) \cap W$. Define $\mathcal{N} = \tilde{T} \cup \bigcup_{R \in \mathcal{R}} N_R$.

We now restrict B in W to use only vertices in the neighborhood that have been neighbors in a solution in $\mathcal{R}$, that is, only vertices in $\mathcal{N}$. Define $\hat{B} = (B \cap U) \cup (B \cap \mathcal{N})$. We now replace B in our instance with $\hat{B}$ and apply all our reduction rules exhaustively to arrive at the instance $(G^*, I^*, O^*, \hat{B}^*, \omega^*, k^*, T^*)$. Finally, we also solve this instance recursively and return the solutions after undoing the reduction rules.

Correctness. We already proved that the reduction rules and the algorithm for the unbreakable case are correct. The main statement we have to show is that we can replace B with $\hat{B}$ without throwing away important solutions. That means that the instances $(G, I, O, B, \omega, k, T)$ and $\hat{\mathcal{I}} \coloneqq (G, I, O, \hat{B}, \omega, k, T)$ are equivalent in the sense that any solution set for one instance can be transformed to a solution set to the other instance with at least the same weights. This justifies solving $\hat{\mathcal{I}}$ instead of $\mathcal{I}$. Consider the boundary complementation $\mathcal{I}' \coloneqq (G', I', O', B', \omega', k')$ and $\hat{\mathcal{I}}' \coloneqq (G', I', O', \hat{B}', \omega', k')$ that are caused by the same (X, Y, Z, R). To show the claim, we consider the two directions. Any solution for $\hat{\mathcal{I}}'$ is immediately a solution for the same boundary complementation for $\mathcal{I}'$ since the only difference is that we limit the possible neighborhood to a subset of B.

For the other direction, consider a solution S to $\mathcal{I}'$, and we want to show that there is a solution $\hat{S}$ to $\hat{\mathcal{I}}'$ using only vertices of $\hat{B}'$ in the neighborhood with $\omega(\hat{S}) \geq \omega(S)$. If $S \cap W = \emptyset$, then S is also immediately a solution for $\hat{\mathcal{I}}'$. Therefore, assume $S \cap W \neq \emptyset$. Define $\tilde{X} \coloneqq \tilde{T} \cap S$, $\tilde{Y} \coloneqq \tilde{T} \cap \mathrm{N}_{G-(W \setminus U)}(S)$, and $\tilde{Z} \coloneqq \tilde{T} \setminus (\tilde{X} \cup \tilde{Y})$. Let $\tilde{R}$ be the set of $(a, b) \in \tilde{X} \times \tilde{X}$ such that there is an a-b-path in $G[S \setminus W \setminus U]$. Finally, let $\tilde{k} \coloneqq |\mathrm{N}(S) \cap W|$. Thus, we can construct a boundary complementation instance $(\tilde{G}', \tilde{I}', \tilde{O}', \tilde{B}', \tilde{w}', \tilde{k})$ from $(\tilde{X}, \tilde{Y}, \tilde{Z}, \tilde{R})$. One can easily verify that $(S \cap \mathrm{V}(\tilde{G}')) \cup \left\{ u_r \;\middle|\; r \in \tilde{R} \right\}$ is a feasible solution to this instance. Furthermore, the maximum solution $\tilde{S} \in \mathcal{R}$ to this instance gives a new set $\hat{S} \coloneqq (\tilde{S} \cap W) \cup (S \cap U) \subseteq \mathrm{V}(G')$ that has at least the same weight as S. We also know that $\mathrm{N}(\hat{S}) \subseteq \hat{B}'$ and $\left|\mathrm{N}(\hat{S})\right| \leq k$. See Figure 2 for a visualization of the construction corresponding to a solution S.

All that remains is to verify that $\hat{S}$ is strongly connected. For $v_1, v_2 \in \tilde{S} \cap W$, we can simply use the v_1-v_2-path in $\tilde{S}$, replacing subpaths via u_r for $r \in \tilde{R}$ with the actual paths in $S \cap U$. If $v_1 \in \tilde{S} \cap W$ and $v_2 \in S \cap U$, we can walk to any $x \in \tilde{X}$ which must have a path to v_2 in S, in which may need to replace subpaths via W. This can be done, since $\tilde{S} \cap W$ connects all pairs of $x_1, x_2 \in X$ that are not connected via $S \cap U$. The two remaining cases follow by symmetry, justifying the replacement of B with $\hat{B}$.

Finally, we have to show that the recursion terminates for the right choice of c; that is, both recursively solved instances have strictly smaller sizes than the original graph. For the first recursive call, note that the boundary complementation adds at most $k^2 \leq q$ vertices to $\tilde{G}$. Since $|U \setminus W| > q$, we have $\left|\mathrm{V}(\tilde{G})\right| < |\mathrm{V}(G)|$.

For the second recursive call, since $\left|\tilde{T}\right| \leq 2k$ and by Lemma 9, we have $|\mathcal{N}| \leq 2k + (k+1)k2^{c_1 k^2} \leq 2^{c_2 k^2}$ for constants c_1 and c_2. After applying all our reduction rules, by Lemma 25 for a suitable choice of c_3 and c we get

$$|W^*| \leq |\mathcal{N}| + 2^{2^{|\mathcal{N}|+1}+|\mathcal{N}|^2}\left(2^{|\mathcal{N}|+1} + |\mathcal{N}|^2\right) \leq 2^{c_3 2^{|\mathcal{N}|}} \leq 2^{c_3 2^{2^{c_2 k^2}}} \leq 2^{2^{2^{ck^2}}} =: q.$$

Since before the reductions, we had $|W \setminus U| > q$, the reduced graph G^* also has fewer vertices than G. Therefore, this recursive call also makes progress and the recursion terminates.

Runtime. We follow the analysis of [3]. With Lemma 26, we can test if the undirected version of G is unbreakable in time $2^{k \log q + k} n^{\mathcal{O}(1)} \leq 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. If the graph turns out to be unbreakable, the algorithm of Theorem 12 solves it in time $2^{\mathcal{O}(k^2 \log q)} \leq 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. Executing the reduction rules takes polynomial time in n by Lemma 25.

All that is left to do is analyze the recursion. Let $n' \coloneqq |W|$. By the separation property, we know that $q < n' < n - q$. From the correctness section, we get $|\mathrm{V}(G^*)| \leq |\mathrm{V}(G)| - n' + q$. Note that the recursion stops when the original graph is (q,k)-unbreakable, which must be the case if $|\mathrm{V}(G)| \leq 2q$. We arrive at the recurrence

$$T(n) = \begin{cases} 2^{2^{2^{\mathcal{O}(k^2)}}}, & \text{for } n \leq 2q; \\ \left(\max_{q<n'<n-q} T(n'+k^2) + T(n-n'+q)\right) + 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}, & \text{otherwise.} \end{cases}$$

Notice that $2^{2^{2^{\mathcal{O}(k^2)}}}$ appears in every summand of the expanded recurrence and can be ignored here and multiplied later. Furthermore, $n^{\mathcal{O}(1)}$ is bounded from above by a convex polynomial. Therefore, it is enough to consider the extremes of the maximum expression. For $n' = q+1$, the first recursive call evaluates to $T(q+1+k^2) \leq T(2q) \leq 2^{2^{2^{\mathcal{O}(k^2)}}}$. The second call only introduces an additional factor of n. For $n' = n-q-1$, the second expression evaluates to $T(2q+1)$ which is clearly also bounded by $2^{2^{2^{\mathcal{O}(k^2)}}}$. Thus, we arrive at the final runtime of $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. ◀

Finally, we can use Boundaried Max TSSCS to solve TSSCS. Since in our original problem every vertex could be part of the solution or its neighborhood, we initially set $I \coloneqq O \coloneqq \emptyset$ and $B \coloneqq \mathrm{V}(G)$. Furthermore, we only want to consider the boundary complementation that changes nothing, which we achieve by setting $T \coloneqq \emptyset$.

▶ **Theorem 1.** *Total-Secluded Strongly Connected Subgraph is solvable in time* $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$.

3 Secluded $\mathcal{F}$-Free subgraph and Secluded DAG

In this section, we show that Out-Secluded $\mathcal{F}$-Free Subgraph is W[1]-hard for almost all choices of $\mathcal{F}$, even if we enforce weakly connected solutions. Except for a few missing cases, we establish a complete dichotomy when Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph (Out-Secluded $\mathcal{F}$-Free WCS) is hard and when it is FPT. This is a surprising

result compared to undirected graphs and shows that out-neighborhood behaves completely different. The same problem was studied before in undirected graphs, where it admits FPT-algorithms using recursive understanding [11] and branching on important separators [15]. For total neighborhood, both algorithms still work efficiently since the seclusion condition acts exactly the same as in undirected graphs. However, the result changes when we look at out-neighborhood, where we show hardness in Theorem 2 for almost all choices of $\mathcal{F}$. To formalize for which kind of $\mathcal{F}$ the problem becomes W[1]-hard, we need one more definition.

▶ **Definition 28** (Inward Star). *We say that a directed graph G is an* inward star*, if there is one vertex $v \in \mathrm{V}(G)$ such that $\mathrm{E}(G) = \{(u, v) \mid u \in \mathrm{V}(G) \setminus \{v\}\}$, that is, the underlying undirected graph of G is a star and all edges are directed towards the center.*

Due to the nature of the construction, we show that if no inward star contains any $F \in \mathcal{F}$ as a subgraph, the problem becomes W[1]-hard. Besides this restriction, the statement holds for all non-empty $\mathcal{F}$, even families with only a single forbidden induced subgraph. Later, we explain how some other cases of $\mathcal{F}$ can be categorized as FPT or W[1]-hard.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a non-empty set of directed graphs such that no $F \in \mathcal{F}$ is a subgraph of an inward star. Then, Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph is W[1]-hard with respect to the parameter $k + w$ for unit weights.*

Proof Sketch. Let $F \in \mathcal{F}$. We give a reduction from Clique, inspired by [9]. Given an undirected graph G, k, and w, consider the incidence graph, which we modify in the following. We add $k + 2$ copies of F and a single vertex s. For every edge $e = \{u, v\} \in E$, we add the edges (e, u), (e, v), and (e, s). Furthermore, we add an edge from every vertex $v \in V_V$ to every vertex in every copy of F. For two different copies F_1 and F_2 of F, we add edges in both directions between every vertex $v_{f1} \in F_1$ and v_{f2} in F_2. Denote this newly constructed graph by G'. Finally, set $k' \coloneqq k$ and $w \coloneqq \binom{k}{2} + 1$.

The proof follows by showing that a clique of size k corresponds exactly to an out-secluded F-free weakly connected subgraph in G' with parameters k' and w. More details can be found in the full version [21]. ◀

Note that Theorem 2 is very general and excludes many properties Π immediately from admitting FPT-algorithms. Another interesting example is finding secluded DAGs, a very natural extension of secluded trees, studied in [11, 7]. There we choose $\mathcal{F}$ to be the set of all directed cycles. Although this set is not finite, hardness follows from Theorem 2.

▶ **Corollary 3.** *Out-Secluded Weakly Connected DAG is W[1]-hard with parameter $k + w$ for unit weights.*

Finally, we analyze some of the more remaining cases in the following paragraphs.

Trivial Cases. If $\mathcal{F}$ contains the graph with only a single vertex, the empty set is the only $\mathcal{F}$-free solution. If $\mathcal{F}$ contains two vertices connected by a single edge, weakly connected solutions can only consist of a single vertex or contain bidirectional edges (u, v) and (v, u) for $u, v \in \mathrm{V}(G)$. Both of these problems are clearly solvable in FPT-time, the second one via our algorithm for Secluded Clique. Additionally, if $\mathcal{F} = \emptyset$, we can clearly choose the maximum weight component of G as the solution.

Independent Sets. An independent set is a subgraph of an inward star. One kind of graph that cannot be included in $\mathcal{F}$ such that Theorem 2 shows hardness are independent sets. Surprisingly, the problem becomes FPT if $\mathcal{F}$ includes an independent set of any size. First, notice that independent sets that are part of $\mathcal{F}$ can be ignored if there is a smaller independent set in $\mathcal{F}$. Suppose the size of the smallest independent set is $\alpha + 1$. This means that any solution must be an α-bounded graph, i.e., a graph without an independent set of size α.

We have shown already how these can be enumerated efficiently with the algorithm in the proof of Theorem 4. For every enumerated subgraph, we can simply check if it is $\mathcal{F}$-free in time $|\mathcal{F}|\, n^{\|\mathcal{F}\|}$. Therefore, this case is also FPT with parameter k if $\mathcal{F}$ is finite.

▶ **Theorem 29.** *Let $\mathcal{F}$ be a finite family of directed graphs that contains an independent set. Then* Out-Secluded $\mathcal{F}$-Free WCS *is solvable in time $|\mathcal{F}|\,(2\alpha+2)^k n^{\|\mathcal{F}\|+\alpha+\mathcal{O}(1)}$.*

Inward Stars. Consider what happens when $\mathcal{F}$ contains an inward star F with two leaves. Then, the weakly connected F-free graphs are exactly the rooted trees where the root could be a cycle instead of a single vertex, with potentially added bidirectional edges. For $F \in \mathcal{F}$, we do not have a solution, but we conjecture that the FPT branching algorithm for Secluded Tree by [7] can be transferred to the directed setting.

If $\mathcal{F} = \{F\}$ for a star F with more than two leaves, the problem remains W[1]-hard. We can slightly modify the construction in the proof of Theorem 2 such that there is not one extra vertex s, but one extra vertex s_e for every $e \in \mathrm{E}(G)$. We connect all s_e internally in a directed path. A solution for Out-Secluded $\mathcal{F}$-Free WCS can then include the whole extra path and the edges encoding the clique. This avoids inward star structures with more than two leaves, showing hardness in this case. For only one single forbidden induced subgraph F, we therefore conjecture, that Out-Secluded $\mathcal{F}$-Free WCS is FPT with parameter k if and only if F has no edges or is an inward star with at most two leaves. For all other cases, we have proven W[1]-hardness with parameter $k + w$ for unit weights.

Remaining Cases. The previous paragraphs resolve almost all possible cases for $\mathcal{F}$, except for a few cases which are difficult to characterize. For example, we could have $\{F_s, F_p\} \subseteq \mathcal{F}$, where F_s is an inward star and F_p is a path, which makes a new connecting construction instead of s and the s_e necessary. The above characterization is still enough to give an understanding of which cases are hard and which are FPT for all natural choices of $\mathcal{F}$.

4 Secluded Subgraphs with Small Independence Number

In this section, we consider a generalization of the undirected Secluded Clique problem to directed graphs. We generalize this property in the following way.

▶ **Definition 30** (α-Bounded). *A directed graph G is called* α-bounded *if G includes no independent set of size $\alpha + 1$, that is, for all $S \subseteq \mathrm{V}(G)$ with $|S| = \alpha + 1$, the graph $G[S]$ includes at least one edge.*

Our problem of interest will be the Out-Secluded α-Bounded Subgraph problem (Out-Secluded α-BS). Note that α is part of the problem and therefore a constant. In the directed case, $\alpha = 1$ is analogous to the Out-Secluded Tournament problem, except that tournaments cannot include more than one edge between a pair of vertices.

In this section, we show how to solve Out-Secluded α-BS with a branching algorithm. The general nature of these branching rules allows us to transfer and optimize them, to significantly improve the best known runtime for the Secluded Clique problem to $1.6181^k n^{\mathcal{O}(1)}$ in the full version.

Our branching algorithm starts by selecting one part of the solution and then relies on the fact that the remaining solution has to be close around the selected part. More concretely, we start by picking an independent set that is part of the solution and build the remaining solution within its two-hop neighborhood. The following lemma justifies this strategy. The proof of this lemma was sketched on *Mathematics Stack Exchange* [14], we give an inspired proof again for completeness.

▶ **Lemma 31.** *In every directed graph G, there is an independent set $U \subseteq \mathrm{V}(G)$, such that every vertex in $\mathrm{V}(G) \setminus U$ is reachable from an $u \in U$ via a path of length at most 2.*

Proof. We prove the statement by induction on $|\mathrm{V}(G)|$. For $|\mathrm{V}(G)| = 1$, it clearly holds. Assume the statement holds for all graphs with fewer than $|\mathrm{V}(G)|$ vertices, we want to prove it for G. Let $v \in \mathrm{V}(G)$ be a vertex. If v reaches every other vertex via at most 2 edges, $\{v\}$ is our desired independent set. Otherwise, $T \coloneqq \mathrm{V}(G) \setminus \mathrm{N}^+[v]$ is non-empty. Since $|T| < |\mathrm{V}(G)|$, we can apply the induction hypothesis. So, let $T' \subseteq T$ be an independent set in $G[T]$ that reaches every vertex of T via at most two edges. We consider the edges between v and T'.

Since $T \cap \mathrm{N}^+(v) = \emptyset$, there cannot be an edge (v, t') for any $t' \in T'$. If there is an edge (t', v) for some $t' \in T'$, then T' is also a solution for G since t' can reach $\mathrm{N}^+[v]$ via at most 2 edges. Finally, if there is no edge between v and T', then $T' \cup \{v\}$ is an independent set. Also $T' \cup \{v\}$ reaches by definition all of $T \cup \mathrm{N}^+[v]$ via at most 2 edges. ◀

For α-bounded graphs, all independent set have size at most α, so trying every subset of $\mathrm{V}(G)$ of size at most α allows us to find a set that plays the role of U in Lemma 31 in our solution. Hence, the first step of our algorithm will be iterate over all such sets. From then on, we only consider solutions $S \subseteq \mathrm{N}^+[\mathrm{N}^+[U]]$.

Then, our algorithm uses two branching rules. Both rules have in common that in contrast to many well-known branching algorithms [5, Chapter 3], we never include vertices into a partial solution. Instead, due to our parameter, we only decide that vertices should be part of the final neighborhood. This means that we remove the vertex from the graph and decrease the parameter by 1. Our first branching rule branches on independent sets of size $\alpha + 1$ in $\mathrm{N}^+[\mathrm{N}^+[U]]$, to ensure that $\mathrm{N}^+[\mathrm{N}^+[U]]$ becomes α-bounded. The second rule is then used to decrease the size of the neighborhood of $\mathrm{N}^+[\mathrm{N}^+[U]]$, until it becomes secluded.

To formalize which vertices exactly to branch on, we use some new notation. For a vertex $v \in \mathrm{V}(G)$ and a vertex set $U \subseteq \mathrm{V}(G)$, pick an arbitrary shortest path from U to v if there exists one. Define $P_U(v)$ to be the vertices on that path including v but excluding the first vertex $u \in U$. Note that after removing vertices from the graph, $P_U(v)$ might change.

Now, we can give our algorithm for finding secluded α-bounded graphs.

▶ **Theorem 4.** *Out-Secluded α-Bounded Subgraph is solvable in time $(2\alpha+2)^k n^{\alpha+\mathcal{O}(1)}$.*

Proof. Let (G, ω, w, k) be an instance of Out-Secluded α-BS. We guess a vertex set $U \subseteq \mathrm{V}(G)$ that should be part of a desired solution S. Furthermore, we want to find a solution S to the instance such that $S \subseteq \mathrm{N}^+[\mathrm{N}^+[U]]$, that is, every $v \in S$ should be reachable from U via at most two edges. We give a recursive branching algorithm that finds the optimal solution under these additional constraints. The algorithm is also described in Algorithm 2.

Algorithm 2 The branching algorithm for Out-Secluded α-BS that returns a solution including the set $U \subseteq V(G)$.

```
def α-BS(G, ω, w, k, U):
    if k < 0 then
        abort
    else if N⁺[N⁺[U]] is a solution then
        return N⁺[N⁺[U]]
    else if there is an independent set I ⊆ N⁺[N⁺[U]] of size |I| = α + 1 then
        foreach v ∈ ⋃_{w∈I} P_U(w) do
            Call α-BS(G − v, ω, w, k − 1, U)
        end
    else if there is w ∈ N⁺(N⁺[N⁺[U]]) then
        foreach v ∈ P_U(w) do
            Call α-BS(G − v, ω, w, k − 1, U)
        end
    end
```

When deciding that a vertex should be part of the final neighborhood, we can simply delete it and decrease k by one. In case k decreases below 0, there is no solution. If $N^+[N^+[U]]$ is a solution to the instance, we return it. Otherwise we apply the following branching rules and repeat the algorithm for all non-empty independent sets $U \subseteq V(G)$ of size at most α.

Case 1. $N^+[N^+[U]]$ is not α-bounded. In this case, there must be an independent set $I \subseteq N^+[N^+[U]]$ of size $\alpha + 1$. Clearly, not all of I can be part of the solution, so there is a vertex $w \in I \setminus S$. This means that either w or a vertex on every path from U to w must be in $N^+(S)$. The set $P_U(w)$ is one such path of length at most 2. Thus, one of $\bigcup_{w \in I} P_U(w)$ must be part of the out-neighborhood of S and we branch on all of these vertices. For one vertex, delete it and decrease k by 1.

Case 2. $N^+[N^+[U]]$ is α-bounded, but has additional neighbors. Consider one of these neighbors $w \in N^+(N^+[N^+[U]])$. Since w is not reachable from U via at most two edges, we should not include it in the solution. Now, $P_U(w)$ is a path of length 3, and one of its vertices must be in $N^+(S)$. We again branch on all options, delete the corresponding vertex and decrease k by 1.

By Lemma 31, there is a suitable choice of U for every solution. Therefore, if we can find the maximum solution containing U if one exists in every iteration with our branching algorithm, the total algorithm is correct. Also notice that the branching rules are a complete case distinction; if none of the rules apply, the algorithm reaches a base case. The remaining proof of correctness follows from a simple induction.

We initially have to consider all subsets of $V(G)$ of size at most α while rejecting subsets that are not an independent set in time $n^{\alpha+1}$. To bound the runtime of the branching algorithm, notice that in each case we branch and make progress decreasing k by 1. In the first cases, the independent set I has size $\alpha + 1$ for every $w \in I$, the path $P_U(w)$ includes at most 2 vertices. Hence, there are at most $2(\alpha + 1)$ branches in this case. The second case gives exactly 3 branches and is thus dominated by the first rule. This gives the claimed runtime and concludes the proof. ◀

References

1 Shiri Chechik, Matthew P Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79:708–741, 2017. doi:10.1007/s00453-016-0222-z.

2 Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55:1–19, 2008. doi:10.1145/1411509.1411511.

3 Rajesh Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing fpt algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45:1171–1229, 2016. doi:10.1137/15M1032077.

4 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM Journal on Computing*, 42:1674–1696, 2013. doi:10.1137/12086217X.

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 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48:417–450, 2019. doi:10.1137/140988553.

7 Huib Donkers, Bart M.P. Jansen, and Jari J.H. de Kroon. Finding k-secluded trees faster. *Journal of Computer and System Sciences*, 138:103461, 2023. doi:10.1016/j.jcss.2023.05.006.

8 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory of Computing Systems*, 61:795–819, 2017. doi:10.1007/s00224-016-9717-x.

9 Fedor V Fomin, Petr A Golovach, and Janne H Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *Mathematical Foundations of Computer Science*, pages 421–432, 2013. doi:10.1007/978-3-642-40313-2_38.

10 Alexandra Fradkin and Paul Seymour. Edge-disjoint paths in digraphs with bounded independence number. *Journal of Combinatorial Theory, Series B*, 110:19–46, 2015. doi:10.1016/j.jctb.2014.07.002.

11 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. *Journal of Computer and System Sciences*, 113:101–124, 2020. doi:10.1016/j.jcss.2020.05.006.

12 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

13 Meike Hatzel, Lars Jaffke, Paloma T Lima, Tomáš Masařík, Marcin Pilipczuk, Roohani Sharma, and Manuel Sorge. Fixed-parameter tractability of directed multicut with three terminal pairs parameterized by the size of the cutset: twin-width meets flow-augmentation. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3229–3244, 2023. doi:10.1137/1.9781611977554.ch123.

14 Sera Gunn (https://math.stackexchange.com/users/437127/sera gunn). Independent set of vertices in a directed graph. Mathematics Stack Exchange, 2017. URL: https://math.stackexchange.com/q/2292101.

15 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Single-exponential FPT algorithms for enumerating secluded f-free subgraphs and deleting to scattered graph classes. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, pages 42:1–42:18, 2023. doi:10.4230/LIPIcs.ISAAC.2023.42.

16 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO Model Checking to Highly Connected Graphs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 135:1–135:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

17 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length-and neighborhood-constrained path problems. *Information Processing Letters*, 156:105913, 2020. doi:10.1016/j.ipl.2019.105913.

18 Pranabendu Misra, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Sub-exponential time parameterized algorithms for graph layout problems on digraphs with bounded independence number. *Algorithmica*, 85:2065–2086, 2023. doi:10.1007/s00453-022-01093-w.

19 Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Transactions on Computation Theory*, 10:1–18, 2018. doi:10.1145/3201775.

20 Abhishek Sahu and Saket Saurabh. Kernelization of arc disjoint cycle packing in α-bounded digraphs. *Theory of Computing Systems*, 67:221–233, 2023. doi:10.1007/s00224-022-10114-8.

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

22 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondřej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discrete Optimization*, 30:20–50, 2018. doi:10.1016/j.disopt.2018.05.002.

23 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.