Solution Discovery for Vertex Cover, Independent Set, Dominating Set, and Feedback Vertex Set*

Rin Saito[1], Anouk Sommer[2], Tatsuhiro Suga[1], Takahiro Suzuki[1], and Yuma Tamura[1]

[1] Graduate School of Information Sciences, Tohoku University, Sendai, Japan
{rin.saito,suga.tatsuhiro.p5,takahiro.suzuki.q4}@dc.tohoku.ac.jp, tamura@tohoku.ac.jp
[2] Karlsruhe Institute of Technology, Germany
anouk.sommer@student.kit.edu

Abstract. In the solution discovery problem for a search problem on graphs, we are given an initial placement of k tokens on the vertices of a graph and asked whether this placement can be transformed into a feasible solution by applying a small number of modifications. In this paper, we study the computational complexity of solution discovery for several fundamental vertex-subset problems on graphs, namely Vertex Cover Discovery, Independent Set Discovery, Dominating Set Discovery, and Feedback Vertex Set Discovery. We first present XP algorithms for all four problems parameterized by clique-width. We then prove that Vertex Cover Discovery, Independent Set Discovery, and Feedback Vertex Set Discovery are NP-complete for chordal graphs and graphs of diameter 2, which have unbounded clique-width. In contrast to these hardness results, we show that all three problems can be solved in polynomial time on split graphs. Furthermore, we design an FPT algorithm for Feedback Vertex Set Discovery parameterized by the number of tokens.

Keywords: solution discovery · graph algorithm · parameterized complexity.

1 Introduction

Many conventional problems in algorithmic theory ask for a feasible solution from a given graph. The feasible solution corresponds to a configuration of a real-world system, such as the placement of surveillance robots in a museum or factory. However, in practical scenarios, a current configuration of a system often already exists, and the task is to obtain a feasible one starting from it. In particular, the current configuration may become infeasible due to system failures and must be restored to a feasible configuration as soon as possible.

To capture this type of situation, Fellows et al. proposed the framework of *solution discovery* [9], and recently, several researchers have studied it in various problems [12,13]. In this framework, the problem Π-Discovery is defined with respect to a combinatorial problem Π together with the upper bound of *modification steps*, which specifies how many steps are allowed to modify a given current configuration into a feasible one. Formally, given an instance of Π, an initial configuration S, and a budget b, Π-Discovery asks whether S can be transformed into a feasible solution to Π within b modification steps. Note that neither the initial nor the intermediate configurations are required to be feasible. Most studies of solution discovery focus on vertex-subset problems on graphs, such as Vertex Cover and Independent Set, and an atomic modification step follows the *token sliding model* [20]. [3] In the token sliding model of Π-Discovery, a vertex subset is regarded as a set of tokens placed on vertices, where each vertex is occupied by at most one token. Then, at each step, one token may slide along an edge. Recalling the example of placing surveillance robots, this model naturally represents the movement of robots between adjacent locations.

* This work was partially supported by JST SPRING Grant Number JPMJSP2114, the *Baden-Württemberg-STIPENDIUM* from the Baden-Württemberg-Stiftung, and JSPS KAKENHI Grant Number JP25K21148.

[3] We use the term "token sliding" following the terminology in the paper by Fellows et al. [9] that introduced the solution discovery framework.

Table 1. The parameterized complexity and complexity on graph classes for VC-D, IS-D, DS-D, and FVS-D. The parameter k denotes the number of tokens. The characters "h" and "c" represent "hard" and "complete", respectively.

		Problem			
		VC-D	FVS-D	IS-D	DS-D
Parameter	k	FPT [9]	FPT [Thm. 2]	W[1]-h [9]	W[1]-h [9]
	clique-width	XP [Thm. 1]	XP [Thm. 1]	XP [Thm. 1]	XP [Thm. 1]
	pathwidth	XNLP-h [13]	XNLP-h	XNLP-h [13]	XNLP-h [13]
Graph Class	chordal	NP-c [Thm. 4]	NP-c [Cor. 1]	NP-c [Cor. 1]	NP-c [18]
	diameter 2	NP-c [Cor. 2]	NP-c [Cor. 2]	NP-c [Cor. 2]	NP-c [18]
	split	P [Thm. 5]	P [Cor. 3]	P [Cor. 3]	NP-c [18]

1.1 Our Contribution

This paper studies the computational complexity of Vertex Cover Discovery, Independent Set Discovery, Dominating Set Discovery, and Feedback Vertex Set Discovery (VC-D, IS-D, DS-D, and FVS-D, respectively) under the token sliding model. An overview of our results is provided in Table 1.[4]

The first three problems have been widely studied in previous work from the viewpoint of parameterized complexity [9,13]. We first show that VC-D, IS-D, and DS-D all admit XP algorithms when parameterized by clique-width. Consequently, these problems can be solved in polynomial time on graphs of bounded clique-width, including graphs of bounded treewidth [6], cographs, and distance-hereditary graphs. In particular, this result extends the earlier result of Fellows et al. that provided XP algorithms for VC-D, IS-D, and DS-D parameterized by treewidth [9]. We emphasize that this result is the best possible for clique-width: Grobler et al. proved the XNLP-hardness for these problems parameterized by pathwidth [13], which implies that no FPT algorithm exists for clique-width unless FPT = W[1], since graphs with bounded pathwidth have bounded treewidth, and then bounded clique-width [6].

We next turn to graph classes of unbounded clique-width, in particular, chordal graphs and their subclass, split graphs. We first observe that DS-D remains NP-complete even on split graphs. This follows immediately from the fact that Dominating Set is NP-complete on split graphs [2]. Remarkably, we prove that VC-D and IS-D are also NP-complete on chordal graphs, even though their underlying problems Vertex Cover and Independent Set are solvable in linear time on chordal graphs [11,23]. To complement this hardness result, we further show that VC-D and IS-D can be solved in polynomial time on split graphs. From the positive results on split graphs (excluding DS-D) and cographs, which have small diameter, a natural research direction is to design polynomial-time algorithms for graphs of small diameter. However, we observe that VC-D, IS-D, and DS-D are NP-complete even for graphs of diameter 2.

In addition, this paper initiates the study of FVS-D. FVS is one of the most fundamental vertex-subset problems, appearing in the list of Karp's 21 NP-complete problems [17], and has been extensively studied in parameterized complexity (see, e.g., [7]). Following the approach for VC-D, we provide an XP algorithm for FVS-D parameterized by clique-width and a polynomial-time algorithm on split graphs, whereas FVS-D is shown to be NP-complete on chordal graphs or graphs of diameter 2. An interesting question is whether FVS-D admits an FPT algorithm when parameterized by the number of tokens k, since VC-D, IS-D, and DS-D are W[1]-hard when parameterized by the feedback vertex set number [13], whereas VC-D is known to be fixed-parameter tractable when parameterized by k [9]. Note that the FPT algorithm for VC-D in [9] crucially relies on enumerating all minimal vertex covers of size at most k. This strategy does not carry over to FVS-D, since the number of minimal feedback vertex sets of size at most k can be as large as $O(n^k)$, and hence such enumeration cannot be done within FPT time. To overcome this obstacle, we exploit the concept of *k-compact representations* of minimal feedback vertex sets [14], which yields a single-exponential FPT algorithm parameterized by k.

[4] The XNLP-hardness of FVS-D parameterized by pathwidth follows by a simple reduction from VC-D; see Section 3.

Related Work Fellows et al. initiated the study of a solution discovery framework for several classical NP-complete graph problems, including VC, IS, and DS, under the token sliding model [9]. They analyzed the parameterized complexity with respect to various parameters, such as the configuration size k and the budget b, and also considered it using notions of graph sparsity. In subsequent work, Grobler et al. investigated the kernelization complexity of VC-D, IS-D, DS-D, and other problems under the token sliding model, and provided a refined classification with respect not only to k and b, but also to structural parameters (e.g., pathwidth and feedback vertex set number). In another line of research, Grobler et al. investigated solution discovery for polynomial-time solvable problems [12].

Very recently, Bousquet et al. [4] provided a meta-algorithm for solution discovery based on model checking of monadic second-order logic (MSO). Note that their results do not imply an XP algorithm for clique-width, which is one of our contributions.

The token sliding model originates from research on *combinatorial reconfiguration* problems (see the surveys by van den Heuvel [15] and Nishimura [20]), which ask whether, given two feasible solutions of a combinatorial problem, there is a step-by-step transformation from one into another without losing the feasibility. In contrast, solution discovery does not specify a target solution, and intermediate configurations may be infeasible.

Organization In Section 2, we give preliminaries and define our problems. In Section 3, we present XP algorithms for VC-D, IS-D, DS-D, and FVS-D parameterized by clique-width, and an FPT algorithm for FVS-D parameterized by the number of tokens. In Section 4, we analyze the complexity of VC-D, IS-D, and FVS-D on graph classes of unbounded clique-width, including chordal graphs, split graphs, and graphs of small diameter. The proofs of claims marked $(*)$ can be found in the Appendix.

2 Preliminaries

For a positive integer p, we write $[p] = \{1, 2, \ldots, p\}$. Let $G = (V, E)$ be an undirected graph, and let $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. For a vertex $v \in V(G)$, we define the *open neighborhood* of v as $N(v) = \{u \in V(G) \colon uv \in E(G)\}$, and the *closed neighborhood* as $N[v] = N(v) \cup \{v\}$. For vertices $u, v \in V(G)$, we write $\mathsf{dist}(u, v)$ for the number of edges in a shortest path between u and v. The *diameter* of a connected graph G, denoted by $\mathsf{diam}(G)$, is $\max_{u,v \in V} \mathsf{dist}(u, v)$.

A *vertex cover* of G is a set $C \subseteq V(G)$ such that every edge has an endpoint in C. An *independent set* of G is a set $I \subseteq V(G)$ such that no two vertices in I are adjacent. A *dominating set* of G is a set $D \subseteq V(G)$ such that every vertex of G belongs to D or has a neighbor in D. A *feedback vertex set* of G is a set $F \subseteq V(G)$ such that every cycle of G contains at least one vertex from F.

Given a graph G and an integer k, the problems Vertex Cover (VC), Independent Set (IS), Dominating Set (DS), and Feedback Vertex Set (FVS) ask whether G contains such a set of size at most k (for VC, DS, FVS) or at least k (for IS).

A *split graph* is a graph whose vertex set can be partitioned into a clique and an independent set. A *chordal graph* is a graph in which every cycle of length at least 4 has a chord, i.e., an edge between two non-consecutive vertices of the cycle.

Our problems Let G be a graph. In this work, a *configuration* of G is simply a vertex subset of G. Two configurations S and S' of G are *adjacent* if there exist $x \in S$ and $y \in V(G) \setminus S$ with $xy \in E(G)$ such that $S' = S \cup \{y\} \setminus \{x\}$. (In particular, $|S| = |S'|$ holds.) A configuration S_t is *reachable in ℓ steps* from S_s if there exists a sequence of configurations $\langle S_s = S_0, S_1, S_2, \ldots S_\ell = S_t \rangle$ such that S_{i-1} and S_i are adjacent for all $i \in [\ell]$. We now define four discovery problems: Vertex Cover Discovery (VC-D), Independent Set Discovery (IS-D), Dominating Set Discovery (DS-D), and Feedback Vertex Set Discovery (FVS-D). Given a graph G, an initial configuration S of size k, and a non-negative integer b (called the *budget*), the problem VC-D (IS-D, DS-D, and FVS-D, respectively) asks whether there exists

a configuration S_t that forms a vertex cover (independent set, dominating set, and feedback vertex set, respectively) of G and is reachable from S in at most b steps.

Throughout this paper, we assume that the input graph is connected; otherwise, the problems can be solved independently on each connected component. We also assume $b \leq k \cdot \mathsf{diam}(G)$ because each token can traverse a path of length at most $\mathsf{diam}(G)$. Moreover, when $b = k \cdot \mathsf{diam}(G)$, the problem Π-Discovery is equivalent to the corresponding underlying problem Π: given an instance (G, k) of Π, we construct an instance (G, S, b) of Π-Discovery, where S is an arbitrary vertex-subset of size k and $b = k \cdot \mathsf{diam}(G)$. Since every configuration of size k is reachable from S within b steps, (G, k) is a yes-instance of Π if and only if (G, S, b) is a yes-instance of Π-Discovery.

Finally, note that all four problems are in NP. Therefore, to prove their NP-completeness, it suffices to show their NP-hardness.

3 Parameterized Algorithms

In this section, we investigate the parameterized complexity of VC-D, IS-D, DS-D, and FVS-D.

It is known that VC-D, IS-D, and DS-D are XNLP-hard when parameterized by pathwidth [13], which implies that no FPT algorithm exists for pathwidth unless FPT = W[1]. We further observe that FVS-D is also XNLP-hard when parameterized by pathwidth: given an instance of VC-D, we can construct an equivalent instance of FVS-D by replacing each edge uv of the input graph with a triangle on vertices u, v, e_{uv}. This transformation increases the pathwidth by exactly one (see also the definition of pathwidth in [7]), and one can observe that this yields a parameterized logspace reduction (see the definition in [3]) from VC-D to FVS-D.

In contrast to these hardness results, we show that all four problems admit XP algorithms when parameterized by clique-width. Recall that the clique-width of a graph is always bounded from above by its treewidth and pathwidth. Therefore, our result strictly strengthens the XP algorithm of Fellows et al. [9] for VC-D, IS-D, and DS-D parameterized by treewidth.

Finally, we show that FVS-D admits a single-exponential FPT algorithm when parameterized by the number of tokens k. Note that the one for VC-D is already known, and IS-D and DS-D are W[1]-hard when parameterized by $k + b$ [9].

3.1 XP algorithm parameterized by clique-width

In this subsection, we first present an XP algorithm for VC-D parameterized by clique-width, based on dynamic programming on a tree structure known as a *w-expression*. Then, by applying the same framework, we also show XP algorithms for FVS-D, IS-D, and DS-D parameterized by clique-width.

We now give the definition of the clique-width of graphs. For a positive integer w, a *w-graph* is a graph in which each vertex is assigned a label from $\{1, 2, \ldots, w\}$. We define the following four operations on w-graphs.

Introduce $i(v)$: Create a graph consisting of a single vertex v labeled with $i \in [w]$.
Union $\oplus$: Take the disjoint union of two w-graphs.
Relabel $\rho_{i \to j}$: For distinct $i, j \in [w]$, replace every label i in the graph with j.
Join $\eta_{i,j}$: For distinct $i, j \in [w]$, add all possible edges between vertices labeled i and vertices labeled j.

The *clique-width* of a graph G is the smallest integer w such that G can be constructed from w-graphs by a sequence of these operations. Such a construction can be represented by a rooted tree, where each node corresponds to one of the above operations. This tree is called a *w-expression tree* of G, and its nodes are referred to as *introduce*, *union*, *relabel*, or *join* nodes, respectively. For a node t of the w-expression tree T, let G_t denote the labeled graph associated with the subtree rooted at t. For node t of T and $i \in [w]$, we call a vertex of G_t with label i an *i-vertex*, and U_i^t denotes the set of i-vertices in G_t. It is known that, given a graph G of clique-width cw, one can compute a $(2^{\mathrm{cw}+1} - 1)$-expression of G in $O(|V(G)|^3)$ time [16,21,22]. A w-expression tree is called *irredundant* if, for each edge uv of G, there exists exactly one join node $\eta_{i,j}$ that introduces the edge uv. Moreover, any w-expression tree can be transformed in linear time into an irredundant w-expression tree with $O(n)$ nodes [6]. Hence, in the following, we may always assume without loss of generality that the w-expression tree is irredundant.

Dynamic Programming Let (G, S, b) be an instance of VC-D, where G has n vertices and T is a w-expression tree of G. We perform a bottom-up dynamic programming over T, where the update rules depend on the type of each node in T.

To simplify the description of our dynamic programming, we introduce a hypothetical vertex v^* that can hold an arbitrary number of tokens and is adjacent to every vertex of G_t for each node t of T. Intuitively, the vertex v^* virtually represents all vertices of G "outside" G_t. We then place $k - |V(G_t) \cap S|$ tokens on v^* in the initial configuration. Let G_t^* denote the graph obtained from G_t by adding v^* and connecting it to all vertices of G_t.

For each node t of T, we maintain a Boolean DP table indexed by tuples $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$, where $\mathtt{K}, \mathtt{A}, \mathtt{P}$ are functions such that $\mathtt{K}\colon [w] \to [k]$ and $\mathtt{A}, \mathtt{P}\colon [w] \to [b]$. Here, ℓ denotes the number of token moves used in G_t ($0 \leq \ell \leq b$). For each label $i \in [w]$, $\mathtt{K}(i)$ represents the number of tokens placed on vertices with label i as the final configuration in G_t^*. The values $\mathtt{A}(i)$ and $\mathtt{P}(i)$ indicate, respectively, the numbers of incoming and outgoing additional moves (*absorption* and *projection*) between i-vertices and v^* in G_t^*. Note that, for each node t, the number of possible tuples is at most $b \cdot k^w \cdot b^{2w} \leq n^{5w+2}$ since $k \leq n$ and $b \leq k \cdot \mathsf{diam}(G) \leq n^2$.

For each tuple s, the table entry $c_t[s]$ is set to `true` if and only if there exists a vertex cover C of G_t and a sequence $\mathcal{S}$ of configurations in G_t^* such that (i) $|C \cap U_z^t| = \mathtt{K}(z)$ for every $z \in [w]$; (ii) $\mathcal{S}$ performs exactly ℓ token moves within G_t, with $\ell \leq b$; (iii) exactly $\mathtt{A}(z)$ moves are performed from v^* to the vertices in U_z^t; and (iv) exactly $\mathtt{P}(z)$ moves are performed from the vertices in U_z^t to v^*.

A tuple $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$ is said to be *valid* if it satisfies the following conditions: (a) $\sum_{z \in [w]} \mathtt{K}(z) \leq k$; (b) if $|U_z^t| = 0$ for $z \in [w]$, then $\mathtt{K}(z) = \mathtt{A}(z) = \mathtt{P}(z) = 0$; (c) $\ell \leq b$; and (d) $|S \cap U_z^t| + \mathtt{A}(z) - \mathtt{P}(z) = \mathtt{K}(z) \leq |U_z^t|$ for all $z \in [w]$. Intuitively, condition (a) ensures that the number of tokens in G_t does not exceed k ($= |S|$). Condition (b) states that if U_z^t is empty, then there is no token on z-vertices as a final configuration, and no token moves occur between z-vertices and v^*. Condition (c) guarantees that the total number of token moves occurring within G_t does not exceed the budget b. Condition (d) enforces the consistency of the number of tokens between the initial and final configurations. Note that, since $|S \cap U_z^t|$ and $|U_z^t|$ are already known for all $z \in [w]$, one can check whether a tuple s is valid in $O(w)$ time. At the root node r of T, our algorithm outputs `YES` if and only if there exists a valid tuple s with $c_r[s] = \mathtt{true}$ such that $\mathtt{A}(z) = \mathtt{P}(z) = 0$ for all $z \in [w]$. As we will see later, the DP entry corresponding to any invalid tuple is always assigned the value `false`, since such moves cannot be realized.

Introduce node $i(v)$: Let t be an introduce node of T, and let $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$ be a tuple for t. In this case, the subgraph G_t consists only of a single vertex v. Since both the empty set $\emptyset$ and singleton $\{v\}$ form vertex covers of G_t, we set $c_t[s] = \mathtt{true}$ if and only if s is a valid tuple and $\ell = 0$.

Union node $\oplus$: Let t be a union node of T with children t_1 and t_2, and let $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$ be a tuple for t. As G_t is the disjoint union of G_{t_1} and G_{t_2}, no tokens move between them. Recall that the union of a vertex cover of G_{t_1} and a vertex cover of G_{t_2} forms a vertex cover of G_t. Thus, each value of $\ell, \mathtt{K}(i), \mathtt{A}(i), \mathtt{P}(i)$ ($i \in [w]$) must be the sum of the corresponding values for t_1 and t_2, respectively. Consequently, we set $c_t[s] = \mathtt{true}$ if and only if s is valid and there exist tuples $s_1 = (\ell_1, \mathtt{K}_1, \mathtt{A}_1, \mathtt{P}_1)$ for t_1 and $s_2 = (\ell_2, \mathtt{K}_2, \mathtt{A}_2, \mathtt{P}_2)$ for t_2 such that

1. $c_{t_1}[s_1] = \mathtt{true}$ and $c_{t_2}[s_2] = \mathtt{true}$,
2. $\ell = \ell_1 + \ell_2$, and
3. $\mathtt{K}(z) = \mathtt{K}_1(z) + \mathtt{K}_2(z)$, $\mathtt{A}(z) = \mathtt{A}_1(z) + \mathtt{A}_2(z)$, and $\mathtt{P}(z) = \mathtt{P}_1(z) + \mathtt{P}_2(z)$ for all $z \in [w]$.

Relabel node $\rho_{i \to j}$: Let t be a relabel node of T with child t', and let $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$ be a tuple for t. Since all i-vertices of $G_{t'}$ are relabeled as j-vertices in G_t (i.e., $U_i^t = \emptyset$), we must have $\mathtt{K}(i) = \mathtt{A}(i) = \mathtt{P}(i) = 0$, and the values of i-vertices for t' are merged into those of j-vertices. Consequently, we set $c_t[s] = \mathtt{true}$ if and only if s is valid, $\mathtt{K}(i) = \mathtt{A}(i) = \mathtt{P}(i) = 0$, and there exists a valid tuple $s' = (\ell', \mathtt{K}', \mathtt{A}', \mathtt{P}')$ for t' such that

1. $c_{t'}[s'] = \mathtt{true}$,
2. $\ell = \ell'$,
3. $\mathtt{K}(j) = \mathtt{K}'(i) + \mathtt{K}'(j)$, $\mathtt{A}(j) = \mathtt{A}'(i) + \mathtt{A}'(j)$, $\mathtt{P}(j) = \mathtt{P}'(i) + \mathtt{P}'(j)$, and
4. $\mathtt{K}(z) = \mathtt{K}'(z)$, $\mathtt{A}(z) = \mathtt{A}'(z)$, $\mathtt{P}(z) = \mathtt{P}'(z)$ for all $z \in [w] \setminus \{i, j\}$.

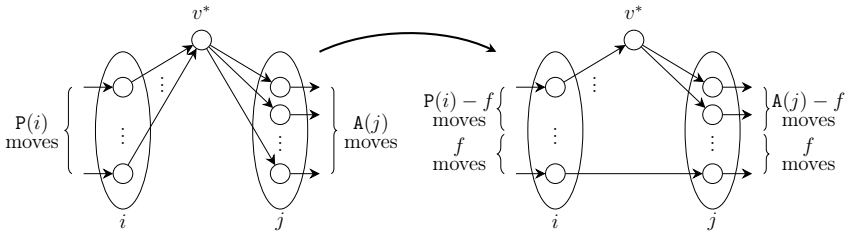

Fig. 1. A rough illustration for a join node. If f tokens directly move from the i-vertices to the j-vertices, we decrease both $\mathtt{P}(i)$ and $\mathtt{A}(j)$ by f, and increase the total number of moves ℓ by f.

Join node $\eta_{i,j}$: Let t be a join node of T with child t', and let $s = (\ell, \mathtt{K}, \mathtt{A}, \mathtt{P})$ be a tuple for t. Since T is irredundant, all edges between i-vertices and j-vertices are added in G_t. Thus, in any vertex cover C of G_t, at least one of the equalities $\mathtt{K}(i) = |U_i^t \cap C|$ or $\mathtt{K}(j) = |U_j^t \cap C|$ must hold; otherwise, some edge between U_i^t and U_j^t is uncovered. Therefore, if both $\mathtt{K}(i) < |U_i^t|$ and $\mathtt{K}(j) < |U_j^t|$, then we set $c_t[s] = \mathtt{false}$.

Otherwise (i.e., $\mathtt{K}(i) = |U_i^t|$ or $\mathtt{K}(j) = |U_j^t|$), we consider the situation that some tokens move between i-vertices and j-vertices. Let $f \geq 0$ denote the number of additional moves from i-vertices to j-vertices, and let $g \geq 0$ denote the number of additional moves in the opposite direction. In other words, we allow tokens to move directly between i-vertices and j-vertices, which were previously treated as moves from i-vertices to j-vertices (or opposite direction), passing through v^*. Therefore, we then set $c_t[s] = \mathtt{true}$ if and only if s is valid and there exist integers $f, g \geq 0$ together with a valid tuple $s' = (\ell', \mathtt{K}', \mathtt{A}', \mathtt{P}')$ for t' such that

1. $c_{t'}[s'] = \mathtt{true}$,
2. $\ell = \ell' + f + g$,
3. $\mathtt{K}(z) = \mathtt{K}'(z)$ for all $z \in [w]$,
4. $\mathtt{A}(z) = \mathtt{A}'(z)$ and $\mathtt{P}(z) = \mathtt{P}'(z)$ for all $z \in [w] \setminus \{i, j\}$, and
5. $\mathtt{A}(i) = \mathtt{A}'(i) - g \geq 0$, $\mathtt{A}(j) = \mathtt{A}'(j) - f \geq 0$, $\mathtt{P}(i) = \mathtt{P}'(i) - f \geq 0$, $\mathtt{P}(j) = \mathtt{P}'(j) - g \geq 0$.

Regarding Conditions 2 and 5, note that there are f tokens that were moving from the i-vertices to the j-vertices via v^*, and g tokens that were moving from the j-vertices to the i-vertices via v^*. Since they originally passed through v^* without incurring any moves, condition 2 makes this correction. In condition 5, we distribute the tokens that were passing through v^* at node t'. A brief illustration is given in Figure 1.

Running time. Observe that the validity of a given tuple s can be verified in $O(w)$ time, and for each type of node t, the value $c_t[s]$ can also be computed within $n^{O(w)}$ time. Since the DP table of a node t contains at most n^{5w+2} entries and the expression tree T has $O(n)$ nodes, VC-D can be solved in $n^{O(w)}$ time.

Remark. The central idea of our algorithm is to perform dynamic programming over a w-expression by guessing the two pieces of information: the feasibility of solutions and the number of token movements. More precisely, we guess the number of tokens on each i-vertex ($i \in [w]$) in the final configuration for VC-D, and furthermore, we maintain auxiliary information that records the number of token movements between i-vertices and v^* in G_t^*, specifically ℓ, $\mathtt{A}$, and $\mathtt{P}$.

In other solution discovery problems, the first part of our approach can be inherited from the well-established dynamic programming approach for the underlying search problem parameterized by clique-width, which has already been known for FVS [1,5], IS, and DS. For instance, to solve FVS-D, one additionally keeps track of information for the connectivity constraints. Consequently, the same dynamic programming technique as VC-D can also be applied to FVS-D, IS-D, and DS-D. This yields the following result.

Theorem 1. *Given a w-expression tree of the input graph, the problems* VC-D, FVS-D, IS-D, *and* DS-D *can be solved in $n^{O(w)}$ time.*

3.2 FPT algorithm parameterized by the number of tokens

This section presents an FPT algorithm for FVS-D parameterized by the number of tokens k.

Theorem 2. FVS-D *parameterized by the number of tokens k can be solved in $O(23.1^k n^3)$ time, where n denotes the number of vertices in the input graph.*

Our algorithm uses an approach for VC-D provided in [9]. We first generate all candidate solutions to FVS. Then, for each candidate solution, construct an edge-weighted complete bipartite graph H and find a minimum weight matching of H saturating one side of H. The key idea is that if one could enumerate all inclusion-wise minimal feedback vertex sets of size at most k, then the above approach yields an FPT algorithm for k. Unfortunately, in general, enumerating minimal feedback vertex sets cannot be done in FPT time with respect to k. For example, consider a graph consisting of k vertex-disjoint cycles of length n/k. The graph contains $\Omega((n/k)^k)$ distinct minimal feedback vertex sets.

We overcome this difficulty by utilizing the concept of *compact representations* of minimal feedback vertex sets [14]. The family $\mathcal{Y} = \{Y_1, Y_2, \dots\}$ of pairwise disjoint vertex subsets of a graph G is a *compact representation* of minimal feedback vertex sets if every set F with $|F \cap Y| = 1$ for all $Y \in \mathcal{Y}$ forms a minimal feedback vertex set of G. In particular, if $|\mathcal{Y}| \leq k$, then $\mathcal{Y}$ is called a *k-compact representation.* We say that $\mathcal{Y}$ *represents* a minimal feedback vertex set F of G if $|F \cap Y| = 1$ for every $Y \in \mathcal{Y}$. A list $\mathfrak{L}$ of k-compact representations is said to be *complete* if, for every minimal feedback vertex set F of size at most k, there is $\mathcal{Y} \in \mathfrak{L}$ that represents F. In other words, every feedback vertex set of size at most k is represented by at least one compact representation in $\mathfrak{L}$.

We use the following result as a subroutine for our algorithm.

Theorem 3 (Theorem 3 of [19]). *Given a graph G with m edges and a positive integer k, there exists an algorithm that computes a complete list of k-compact representations of G in $O(23.1^k m)$ time. Moreover, the number of k-compact representations produced by the algorithm is at most $O(23.1^k)$.*

Let (G, S, b) be an instance of FVS-D. In our algorithm, we first construct a complete list $\mathfrak{L}$ of k-compact representations of minimal feedback vertex sets in G by Theorem 3. Second, for each k-compact representation $\mathcal{Y} \in \mathfrak{L}$, we examine whether there exists at least one feedback vertex set represented by $\mathcal{Y}$ that is reachable from S within b steps.

Now, we explain the second step of our algorithm. Fix a k-compact representation $\mathcal{Y} \in \mathfrak{L}$. We construct an edge-weighted complete bipartite graph $H_{\mathcal{Y}}$ with bipartitions S and $\mathcal{Y}$. To distinguish the vertices of $H_{\mathcal{Y}}$ and the vertices of G, we refer to the vertices of $H_{\mathcal{Y}}$ as *nodes*. For $u \in S$ and $Y \in \mathcal{Y}$, we assign the weight of the edge (u, Y) to $\min_{y \in Y} \mathsf{dist}(u, y)$, which corresponds to the minimum number of steps required to move the token initially placed on u to some vertex in Y. We then consider a minimum-weight matching M^* of $H_{\mathcal{Y}}$ such that each node in $\mathcal{Y}$ of $H_{\mathcal{Y}}$ is incident to some edge in M^*.

The following Lemma 1 establishes the correspondence between the total weight of M and the minimum number of steps to reach some minimal feedback vertex sets represented by $\mathcal{Y}$ from S. The proof can be found in Section A.1.

Lemma 1 ($*$). *The two statements are equivalent:*

(1) *There exists a matching $M = \{(u_1, Y_1), (u_2, Y_2), \dots, (u_{|\mathcal{Y}|}, Y_{|\mathcal{Y}|})\}$ of $H_{\mathcal{Y}}$ with total weight at most b, where $u_i \in S$ and $Y_i \in \mathcal{Y}$ for $i \in [|\mathcal{Y}|]$; and*
(2) *There exists a feedback vertex set F that is reachable in b steps from S and a minimal feedback vertex set $F' \subseteq F$ represented by $\mathcal{Y}$.*

Our algorithm outputs `YES` if there exists a k-compact representation $\mathcal{Y} \in \mathfrak{L}$ such that the corresponding bipartite graph $H_{\mathcal{Y}}$ admits a minimum-weight matching M^* of total weight at most b, and outputs `NO` otherwise.

Finally, we analyze the running time of our algorithm. A complete list $\mathfrak{L}$ of k-compact representations such that $|\mathfrak{L}| = O(23.1^k)$ can be obtained in $O(23.1^k m)$ time [19]. To construct the bipartite graph $H_{\mathcal{Y}}$ for $\mathcal{Y} \in \mathfrak{L}$, we precompute in $O(k(n+m))$ time the distances from every pair of vertices $u \in S$ and $v \in V$

by applying a breadth-first search algorithm k times. For each $\mathcal{Y} \in \mathfrak{L}$, the graph $H_{\mathcal{Y}}$ is constructed in $O(kn)$ time; for each $u \in S$, the weights of the edges incident to u can be decided in $O(n)$ time because $\sum_{\mathcal{Y} \in \mathfrak{L}} |\mathcal{Y}| = O(n)$. Since $H_{\mathcal{Y}}$ has at most $2k$ vertices, the desired minimum-weight matching M^* of $H_{\mathcal{Y}}$ can be obtained in $O(|V(H_{\mathcal{Y}})|^3) = O(k^3)$ time using the Hungarian algorithm [8]. Therefore, the overall running time of our algorithm is $O(23.1^k m + k(n+m) + 23.1^k(kn + k^3)) = O(23.1^k(n^2 + k^3)) = O(23.1^k n^3)$.

4 Graphs of Unbounded Clique-width

In this section, we analyze the computational complexity of VC-D, IS-D, and FVS-D on chordal graphs, split graphs, and graphs of bounded diameter, which have unbounded clique-width. In Section 4.1, we establish the NP-completeness of these problems on chordal graphs and graphs of diameter 2. In Section 4.2, we show the polynomial-time solvability of these problems on split graphs. We remark that DS-D is NP-complete (more precisely, W[2]-hard for the number of tokens k) for split graphs of diameter 2, which follows directly from the known result that DS is NP-complete on those graphs [18].

4.1 NP-completeness

We show the NP-completeness of VC-D, IS-D, and FVS-D for chordal graphs and graphs of diameter 2. We first show the following theorem.

Theorem 4. VC-D *is* NP*-complete for chordal graphs.*

For the proof of Theorem 4, we reduce the Exact Cover by 3-Set problem to VC-D. In Exact Cover by 3-Set, we are given a ground set U of $3n$ elements together with a family $\mathcal{X}$ of three-element subsets of U. The task is to decide whether there exists a subfamily of $\mathcal{X}$ consisting of exactly n sets whose union forms U. It is known that Exact Cover by 3-Set is NP-complete [10].

Let $(U, \mathcal{X})$ be an instance of Exact Cover by 3-Set, where the ground set $U = \{u_1, \ldots, u_{3n}\}$ and the family $\mathcal{X} = \{X_1, \ldots, X_m\}$. We construct a corresponding instance (G, S, b) of VC-D as follows (see Figure 2 for an illustration).

Let $Q = \{q_0, \ldots, q_{3n}\}$ be a set of $3n+1$ vertices. For each $i \in [3n]$, the vertex q_i corresponds to the element $u_i \in U$. Next, let $I = \{v_1, \ldots, v_m\}$ be a set of m vertices, where each v_i corresponds to the set $X_i \in \mathcal{X}$. We then define a split graph G' with $V(G') = Q \cup I$ and $E(G') = \{qq' : q, q' \in Q\} \cup \{v_i q_j : v_i \in I, q_j \in Q, u_j \in S_i\}$. Since I forms an independent set and Q forms a clique in G', it follows that G' is indeed a split graph.

We construct the graph G from G' as follows: for each $v_i \in I$, add the four vertices $v_i^1, v_i^2, v_i^3, v_i^4$ so that $N(v_i^j) = \{v_i\} \cup (N(v_i) \cap Q)$ for each $j \in \{1, 2, 3, 4\}$ in G. Note that each v_i^j is a true twin[5] of v_i in G. That is, the induced subgraph $G[\{v_i, v_i^1, v_i^2, v_i^3, v_i^4\}]$ forms a star rooted at v_i with leaves $v_i^1, v_i^2, v_i^3, v_i^4$, and all these vertices share the same neighbors in Q. We denote this star by A_i. Finally, we set $S = \{v_i^1, v_i^2, v_i^3, v_i^4 : i \in [m]\}$ and $b = 3n + n = 4n$.

We observe that the constructed graph G is chordal, since G is obtained from a split graph G' by adding true twins, and split graphs preserve chordality under the operation of adding true twins. Clearly, the instance (G, S, b) for VC-D is constructed in polynomial time. To complete the polynomial-time reduction, we prove the following lemma. The proof can be found in Section B.1.

Lemma 2 (∗). *An instance* $(U, \mathcal{X})$ *of* Exact Cover by 3-Set *is a yes-instance if and only if an instance* (G, S, b) *of* VC-D *is a yes-instance.*

This completes the proof of Theorem 4.

By the equivalence between VC-D and IS-D, we immediately obtain the NP-hardness of IS-D on chordal graphs. For FVS-D, given an instance of VC-D on a chordal graph G, we construct an equivalent instance of FVS-D by replacing each edge uv of G with a triangle on vertices u, v, and e_{uv}. The resulting graph remains chordal. Thus, we have the following corollary.

[5] Two vertices v, v' are *true twins* if $N[v] = N[v']$.

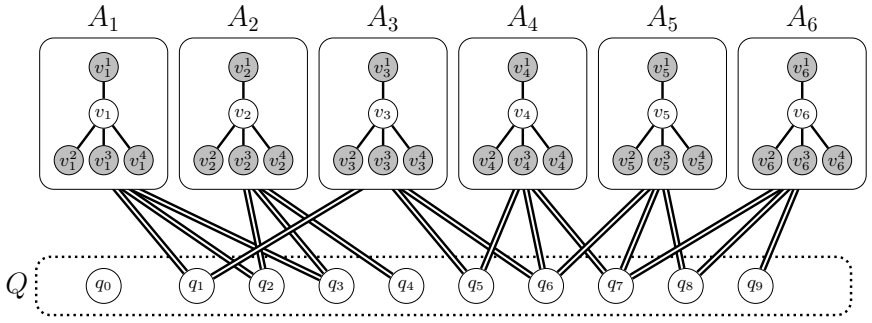

Fig. 2. Construction of an instance (G, S, b) of VC-D from an instance $(U, \mathcal{X})$ of Exact Cover by 3-Set, where $U = \{1, \ldots, 9\}$ and $\mathcal{X} = \{X_1 = \{1, 2, 3\}, X_2 = \{2, 3, 4\}, X_3 = \{1, 5, 6\}, X_4 = \{5, 6, 7\}, X_5 = \{6, 7, 8\}, X_6 = \{7, 8, 9\}\}$. The vertices $q_0, \ldots, q_9$ inside the dotted box form a clique of G. Each double line between a vertex q_i and a solid box labeled A_j represents all edges between q_i and the vertices in A_j. The vertices marked in gray constitute the initial configuration S, and $b = 9 + 3 = 12$.

Corollary 1. IS-D *and* FVS-D *are* NP*-complete for chordal graphs.*

Next, we observe the NP-completeness of VC-D, IS-D, and FVS-D on graphs of diameter 2. Recall that if the budget b equals $k \cdot \mathsf{diam}(G)$, then the problem Π-Discovery is equivalent to the underlying problem Π. Therefore, to prove NP-completeness for VC-D, IS-D, and FVS-D, it suffices to show the NP-completeness of the corresponding underlying problems VC, IS, and FVS on graphs of diameter 2. The details can be found in Section B.2.

Lemma 3 (∗). VC*,* IS*, and* FVS *are* NP*-complete on graphs of diameter* 2*.*

The following Corollary 2 is immediately obtained from Lemma 3.

Corollary 2. VC-D*,* IS-D*, and* FVS-D *are* NP*-complete on graphs of diameter* 2*.*

4.2 Polynomial-time Algorithms

In this subsection, we show that VC-D, IS-D, and FVS-D can be solved in polynomial time on split graphs. We first focus on VC-D and then observe that similar results hold for IS-D and FVS-D. Note that split graphs are a subclass of both chordal graphs and graphs of diameter at most 3. The details can be found in Section B.3.

Theorem 5 (∗). VC-D *can be solved in* $O(n^4)$ *time on split graphs, where* n *is the number of vertices in an input graph.*

To show Theorem 5, we enumerate all minimal vertex covers in a given split graph in polynomial time, and then reduce our problem to the problem of finding a minimum-weight matching, in the same manner as Section 3.2.

We remark that this approach is universal: for any vertex subset problem Π, if one can enumerate candidate solutions of Π or construct a data structure that encodes such information (e.g., compact representations), Π-Discovery can be solved and its running time mainly depends on the enumeration of solutions. In particular, we can observe that maximal independent sets and minimal feedback vertex sets in a split graph can be enumerated in polynomial time. Therefore, we have the following corollary.

Corollary 3. IS-D *and* FVS-D *can be solved in polynomial time on split graphs.*

We also emphasize that Theorem 5 and Corollary 3 can be extended to any graph class in which minimal (maximal for IS-D) solutions can be enumerated in polynomial time.

5 Conclusion

In this paper, we investigated the complexity of VC-D, IS-D, DS-D, and FVS-D from the perspective of parameterization and graph classes. We gave a general scheme for solving Π-Discovery for various vertex-subset problems Π in Section 3.1 and Section 4.2. A promising direction for future research is to identify graph classes for which Π-Discovery is NP-complete, while the underlying problem Π is polynomial-time solvable, as demonstrated for chordal graphs. For interval graphs, which are a subclass of chordal graphs, it is still open whether VC-D, IS-D, DS-D, and FVS-D are solvable in polynomial time. Furthermore, it remains open whether there exists an FPT algorithm for FVS-D parameterized by feedback vertex set number.

Acknowledgements. We thank Akira Suzuki and Xiao Zhou for fruitful discussions. We are also grateful to the anonymous reviewers for their constructive comments, which improved the presentation.

References

1. Bergougnoux, B., Kanté, M.M.: Fast exact algorithms for some connectivity problems parameterized by clique-width. Theoretical Computer Science **782**, 30–53 (2019), https://doi.org/10.1016/j.tcs.2019.02.030
2. Bertossi, A.A.: Dominating sets for split and bipartite graphs. Information Processing Letters **19**(1), 37–40 (1984), https://doi.org/10.1016/0020-0190(84)90126-1
3. Bodlaender, H.L., Groenland, C., Nederlof, J., Swennenhuis, C.: Parameterized problems complete for nondeterministic fpt time and logarithmic space. Information and Computation **300**, 105195 (2024). https://doi.org/https://doi.org/10.1016/j.ic.2024.105195
4. Bousquet, N., Mouawad, A.E., Maaz, S., Nishimura, N., Siebertz, S.: On algorithmic meta-theorems for solution discovery: Tractability and barriers (2025), https://arxiv.org/abs/2510.17344
5. Bui-Xuan, B., Suchý, O., Telle, J.A., Vatshelle, M.: Feedback vertex set on graphs of low clique-width. European Journal of Combinatorics **34**(3), 666–679 (2013), https://doi.org/10.1016/j.ejc.2012.07.023
6. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics **101**(1-3), 77–114 (2000), https://doi.org/10.1016/S0166-218X(99)00184-5
7. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015), https://doi.org/10.1007/978-3-319-21275-3
8. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM **19**(2), 248–264 (1972), https://doi.org/10.1145/321694.321699
9. Fellows, M.R., Grobler, M., Megow, N., Mouawad, A.E., Ramamoorthi, V., Rosamond, F.A., Schmand, D., Siebertz, S.: On solution discovery via reconfiguration. In: ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023). pp. 700–707 (2023), https://doi.org/10.3233/FAIA230334
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
11. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM Journal on Computing **1**(2), 180–187 (1972), https://doi.org/10.1137/0201013
12. Grobler, M., Maaz, S., Megow, N., Mouawad, A.E., Ramamoorthi, V., Schmand, D., Siebertz, S.: Solution discovery via reconfiguration for problems in P. In: 51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia. pp. 76:1–76:20 (2024), https://doi.org/10.4230/LIPIcs.ICALP.2024.76
13. Grobler, M., Maaz, S., Mouawad, A.E., Nishimura, N., Ramamoorthi, V., Siebertz, S.: Kernelization complexity of solution discovery problems. In: 35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia. pp. 36:1–36:17 (2024), https://doi.org/10.4230/LIPIcs.ISAAC.2024.36
14. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. Journal of Computer and System Sciences **72**(8), 1386–1396 (2006), https://doi.org/10.1016/j.jcss.2006.02.001

15. van den Heuvel, J.: The complexity of change. In: Surveys in Combinatorics 2013, London Mathematical Society Lecture Note Series, vol. 409, pp. 127–160. Cambridge University Press (2013). https://doi.org/10.1017/CBO9781139506748.005
16. Hlinený, P., Oum, S.: Finding branch-decompositions and rank-decompositions. SIAM Journal on Computing **38**(3), 1012–1032 (2008), https://doi.org/10.1137/070685920
17. Karp, R.M.: Reducibility among combinatorial problems. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pp. 219–241. Springer (2010), https://doi.org/10.1007/978-3-540-68279-0_8
18. Lokshtanov, D., Misra, N., Philip, G., Ramanujan, M.S., Saurabh, S.: Hardness of r-dominating set on graphs of diameter $(r + 1)$. In: Gutin, G.Z., Szeider, S. (eds.) Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8246, pp. 255–267. Springer (2013), https://doi.org/10.1007/978-3-319-03898-8_22
19. Misra, N., Philip, G., Raman, V., Saurabh, S., Sikdar, S.: FPT algorithms for connected feedback vertex set. Journal of Combinatorial Optimization **24**(2), 131–146 (2012), https://doi.org/10.1007/s10878-011-9394-2
20. Nishimura, N.: Introduction to reconfiguration. Algorithms **11**(4), 52 (2018), https://doi.org/10.3390/a11040052
21. Oum, S.: Approximating rank-width and clique-width quickly. ACM Trans. Algorithms **5**(1), 10:1–10:20 (2008), https://doi.org/10.1145/1435375.1435385
22. Oum, S., Seymour, P.D.: Approximating clique-width and branch-width. Journal of Combinatorial Theory, Series B **96**(4), 514–528 (2006), https://doi.org/10.1016/j.jctb.2005.10.006
23. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM Journal on Computing **5**(2), 266–283 (1976), https://doi.org/10.1137/0205021

A Omitted Proofs in Section 3

A.1 Proof of Lemma 1

We first prove that "(1) $\Rightarrow$ (2)". Suppose that there exists a matching M in $H_{\mathcal{Y}}$ with total weight at most b.

Let $F' = \{v_1, v_2, \ldots, v_{|\mathcal{Y}|}\}$ be a minimal feedback vertex set where, for each $i \in [|\mathcal{Y}|]$, $v_i = \arg\min_{v \in Y_i} \mathsf{dist}(u_i, v)$, that is, v_i is the vertex in Y_i closest to u_i. We then set $F = F' \cup \{u \in S \colon u \text{ is not matched in } M\}$. Note that F is a feedback vertex set with size, as containing F'.

We now claim that F is reachable in b steps from S. To show this, we construct a sequence of token moves with length at most b, as follows: For each $i \in [|\mathcal{Y}|]$, move the token t_i initially placed on u_i along a shortest $u_i v_i$-path. Since the weight of the edge (u_i, Y_i) in $H_{\mathcal{Y}}$ equals $\mathsf{dist}(u_i, v_i)$, we obtain $\sum_{i \in [|\mathcal{Y}|]} \mathsf{dist}(u_i, v_i) \leq b$.

These token moves may involve another token already occupying a vertex on the path. Consider the case where a token t' is initially placed on a vertex u' that lies on the shortest $u_i v_i$-path for some i, with $t' \neq t_i$. If multiple such tokens exist, let t' denote the one closest to u_i along this shortest path. Here, we claim that t' must correspond to some token t_j with index $j \in [|\mathcal{Y}|]$. To prove this, suppose that t' does not correspond to any token t_j with index $j \in [|\mathcal{Y}|]$, i.e., t' is a token that does not move in the constructed sequence. Since $\mathsf{dist}(u_i, v_i) \geq \mathsf{dist}(u', v_i)$, we can replace the matching edge (u_i, v_i) with (u', v_i), thereby obtaining $M' = (M \setminus \{(u_i, v_i)\}) \cup \{(u', v_i)\}$, whose total weight is strictly smaller than that of M. This contradicts the minimality of M.

Thus, we can assume that t' can be expressed as t_j for some $j \in [|\mathcal{Y}|] \setminus \{i\}$. If t_i and t_j are not placed on adjacent vertices, then we simply move t_i toward v_i. Otherwise, i.e., t_i and t_j are placed on adjacent vertices, we swap the indices of t_i and t_j, and continue moving the newly designated t_i toward v_j. A new token t_j is moved to s' at the step when s' is not occupied by any token. Note that t_j cannot reappear on the shortest $u_i v_i$-path before t_i reaches v_i; otherwise, the $u_i v_i$-path would contain a cycle, a contradiction. By repeating the above swapping procedure, t_i eventually reaches v_i. This process does not increase the number of token moves, hence the total length of the sequence remains at most b. Therefore, if there is a matching in $H_{\mathcal{Y}}$ of weight at most b, then there exists a feedback vertex set F that is reachable from S within at most b steps.

We next prove that "(2) $\Rightarrow$ (1)". Suppose that there exists a feedback vertex set F such that a subset $F' \subseteq F$ is represented by $\mathcal{Y}$, and consider the sequence of configurations of length at most b. We show that the corresponding bipartite graph $H_{\mathcal{Y}}$ admits a matching of weight at most b that every vertex in $\mathcal{Y}$ is saturated.

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the set of walks, where each walk P_i describes the moves of a token t_i from its initial position u_i to its target position $g_i \in F$. By construction, the length of each walk does not exceed the length of the shortest $u_i g_i$-path, thus the total length $\sum_{i \in [k]} \mathsf{dist}(u_i, g_i)$ is at most b. Now consider the set $\mathcal{P}' \subseteq \mathcal{P}$ of walks that correspond to the tokens placed on F' in the target configuration.

Since F' is a minimal feedback vertex set of G and F' is represented by $\mathcal{Y}$, there is a matching $M' = \{(u_1, Y_1), (u_1, Y_2), \ldots, (u_{|\mathcal{Y}|}, Y_{|\mathcal{Y}|})\}$ of $H_{\mathcal{Y}}$ such that each Y_i contains the target vertex g_i. Moreover, since each edge (u_i, Y_i) in $H_{\mathcal{Y}}$ is assigned the weight $\min_{c \in Y_i} \mathsf{dist}(u_i, c)$, it follows that the weight of M' is at most b. This completes the proof.

B Omitted Proofs in Section 4

B.1 Proof of Lemma 2

We first show the "only if" direction. Suppose that $(U, \mathcal{X})$ is a yes-instance of Exact Cover by 3-Set, and hence there exists a subfamily $\mathcal{X}' \subseteq \mathcal{X}$ of exactly n sets whose union forms U.

From this subfamily, we construct a vertex cover C that can be reached from S in exactly b steps. For each $X_i \in \mathcal{X}'$ containing elements $u_{j_1}, u_{j_2}, u_{j_3}$ with $j_1, j_2, j_3 \in [3n]$, we move the three tokens initially placed on v_i^2, v_i^3, v_i^4 to the corresponding vertices $q_{j_1}, q_{j_2}, q_{j_3}$ of Q, and slide the token on v_i^1 to v_i. Since this procedure requires four steps for each X_i, the total number of steps is $4n = b$.

By construction, C contains all vertices $q_1, \ldots, q_{3n}$ together with either the root or all leaves of each star A_i, and hence C is indeed a vertex cover of G. Recall that $q_0 \in Q$ is not adjacent to any vertex of the star A_i for every $i \in [m]$. Thus, (G, S, b) is a yes-instance of VC-D.

We next show the "if" direction. Suppose that (G, S, b) is a yes-instance of VC-D, and hence there exists a vertex cover C of G that is reachable from S in b steps.

Since Q is a clique of G with $3n+1$ vertices, any vertex cover of G must contain at least $3n$ vertices from Q. As the initial configuration S places no tokens on Q, moving tokens there requires at least $3n$ steps.

Moreover, to cover the edges of each star A_i for $i \in [m]$, at least one token must be placed on a vertex of A_i. Consequently, at least $3n/3 = n$ stars must contribute at most three tokens to Q, which forces the roots of these n stars to be included in C. Moving tokens to these roots requires at least n additional steps.

Consequently, exactly n stars $A_{i_1}, \ldots, A_{i_n}$ each contribute three tokens to the vertices in Q, while the remaining token in each star is moved to its root. Therefore, the sets $X_{i_1}, \ldots, X_{i_n}$ together cover all elements of U, and hence $(U, \mathcal{X})$ is a yes-instance of Exact Cover by 3-Set.

B.2 Proof of Lemma 3

We first observe that VC remains NP-complete on graphs of diameter 2. Given an instance (G, k) of VC, construct $(G', k+1)$ by adding a universal vertex u adjacent to every vertex in G. Then, we can see that (G, k) is a yes-instance if and only if $(G', k+1)$ is. Thus, Vertex Cover is NP-complete for graphs of diameter at most 2. By the equivalence between VC and IS, IS is also NP-complete for graphs of diameter at most 2.

Next, we show that the claim for FVS is reduced from FVS on general graphs. Given an instance of FVS (G, k), construct $(G', k+1)$ with

$$V(G') = V(G) \cup \{s\} \cup \{v_i^1, v_i^2 : i \in [k+2]\},$$
$$E(G') = E(G) \cup \{sv : v \in V(G)\} \cup \{sv_i^1, sv_i^2, v_i^1 v_i^2 : i \in [k+2]\}.$$

Each triple s, v_i^1, v_i^2 forms a triangle, and s is a universal vertex, so G' has diameter at most 2.

It is straightforward to verify that (G, k) is a yes-instance if and only if $(G', k+1)$ is. Specifically, any feedback vertex set of G' of size at most $k+1$ must include s, and the remaining k vertices form a feedback vertex set of G.

B.3 Proof of Theorem 5

We employ a technique from the algorithm constructed by Fellows et al. for VC-D parameterized by k [9], as well as from our FPT algorithm for FVS-D parameterized by k presented in Section 3.2.

Let $G = (V, E)$ be a split graph that can be partitioned into a clique Q and an independent set I. We assume that each $v \in Q$ has a neighbor in I; otherwise, if $u \in Q$ has no neighbor in I, then we can form another partition with a clique $C \setminus \{u\}$ and an independent set $I \cup \{u\}$. First, we observe that only a small number of token placements C need to be considered. In particular, any minimal vertex cover must place at least $|Q| - 1$ tokens on Q; otherwise, $Q \setminus C$ contains an uncovered edge. If all $|Q|$ tokens are placed on Q, then $C = Q$. If $|Q| - 1$ tokens are placed on Q, there are exactly $|Q|$ choices: for the vertex $v \in Q$ not in C, its neighborhood must be included in C, giving $C = (Q \setminus \{v\}) \cup N(v)$. Thus, the number of minimal vertex covers is exactly $|Q| + 1$, and enumeration can be done in $O(n^2)$ time.

Next, for each minimal vertex cover C, we construct a complete weighted bipartite graph H_C with bipartition (S, C), where the weight of the edge uv is $\mathsf{dist}(u, v)$ in G. We then compute a minimum-weight maximum matching in H_C that matches all vertices in C. If such a matching exists with total weight at most b, then C is reachable from S within b steps, and the algorithm outputs `YES`; otherwise, we proceed to the next cover. If no cover satisfies the condition, the algorithm returns `NO`. Correctness follows the same argument as in Lemma 1.

For the running time, precomputing all-pairs distances takes $O(n^3)$ time. Constructing H_C and solving the problem of finding a minimum-weight maximum matching using the Hungarian algorithm takes $O(n^2 + n^3) =$

$O(n^3)$ for each minimal vertex cover. Since there are $|Q| + 1 = O(n)$ minimal covers, the overall running time is $O(n^3 + n \cdot n^3) = O(n^4)$.