

On Token Movement Problems

by

Stephanie Maaz

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Stephanie Maaz 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Ryuhei Uehara
Professor, Graduate School of Advanced Science and Technology,
Japan Advanced Institute of Science and Technology

Supervisor(s): Naomi Nishimura
Associate Professor, Cheriton School of Computer Science,
University of Waterloo
Amer E. Mouawad
Adjunct Assistant Professor, Cheriton School of Computer Science,
University of Waterloo

Internal Members: Therese Biedl
Professor, Cheriton School of Computer Science,
University of Waterloo
Richard Peng
Adjunct Assistant Professor, Cheriton School of Computer Science,
University of Waterloo

Internal-External Member: Chaitanya Swamy
Professor, Dept. of Combinatorics & Optimization,
University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Classically, the study of computational problems has been primarily focused on what we call a *static model* of the world. Specifically, given a *fixed* unchanging input instance, e.g., a graph, the usual goal is to compute a *fixed* subset of vertices or edges that minimizes or maximizes some objective function. In addition to unchanging instances, the static model ignores the potential existence of prior (partially feasible) solutions as well as the costs associated with materializing a new (more optimal) one. These limitations become more apparent when one considers the *dynamic model* of the world in which we sometimes seek efficient transformations of a given system from some state to another. Upgrading public transport lines is a typical example of this since a reasonable strategy is expected to minimize criteria such as cost, environmental impact, and disruption time. It is therefore crucial for a corresponding computational problem to have knowledge of the current state of the system so as to seek a new, more desirable state, while minimizing the number of required “changes” (and the number of undesired changes).

Solution discovery and *reconfiguration* problems constitute two possible ways of addressing such computational problems arising in our dynamic model of the world, and they are the main topics of this thesis. Both problems model system states as configurations, such as sets of tokens placed on one of vertices or edges of a graph. Under our solution discovery problems, we start with an initial graph configuration and seek to transform it into any final configuration that satisfies a desired property (such as forming a shortest path) and where each vertex contains at most one token, using at most a given budget of token slides along graph edges. In the reconfiguration problems we study, both the initial and target configurations are specified, and we must determine whether one can be transformed into the other within a given budget of token multi-slides, where a token moves along a path of vertices with no other tokens. These token movements capture real-world constraints where changes must be local, that is, prohibiting arbitrary relocation, as seen in applications ranging from robot motion planning to quantum circuit compilation.

This thesis contributes to the computational complexity landscape for these transformation problems. For solution discovery problems where the target property is to form a matching, vertex/edge cut, or shortest path, we show that even though these properties are efficiently computable in the static setting, their transformation variants are NP-hard. Similarly, our reconfiguration problems are NP-hard regardless of whether tokens are distinguishable or indistinguishable. This necessitates a parameterized complexity approach, which provides a more refined analysis by developing (*fixed-parameter tractable*) algorithms whose running times are exponential only in carefully chosen parameters, which are typically small in practice, while remaining polynomial in the input size. Under pa-

parameterized complexity, we investigate these transformation problems, including solution discovery problems for polynomial-time solvable properties (matching, vertex/edge cut, shortest path) and NP-hard properties (independent set, dominating set, vertex cover), under the fundamental parameters of number of tokens k and transformation budget b . Additionally, we examine how certain structural parameters affect the parameterized complexity; particularly, we analyze the independent set, dominating set, and vertex cover solution discovery variants with respect to the parameter pathwidth.

Beyond fixed-parameter tractability, we investigate the *kernelization* complexity of the studied solution discovery problems to understand the limits of preprocessing. Kernelization provides a mathematical framework for analyzing data reduction, asking whether large instances of a problem can be efficiently compressed to equivalent instances whose size depends only on the parameter. A *polynomial kernel* exists when instances of a problem can be reduced to a size that is at most polynomial in the parameter, indicating that effective preprocessing is possible. For problems that we prove admit fixed-parameter tractable algorithms, we employ advanced techniques to establish upper and lower kernel bounds. Our analysis extends to combined parameters, such as examining how the pathwidth structural parameter interacts with the transformation budget to affect preprocessing possibilities.

Finally, we transcend individual problem analysis through meta-theorems that characterize entire families of solution discovery problems using descriptive complexity. Rather than proving similar results repeatedly for each graph property, we investigate general theorems for all solution discovery problems whose target properties are definable in logic, particularly first-order (FO) or the more expressive monadic second-order (MSO) logic. We prove that MSO solution discovery is fixed-parameter tractable when parameterized by the structural parameter neighborhood diversity, exploiting vertex type equivalences to reduce the search space. Conversely, we demonstrate that even FO solution discovery is hard classically and under parameterized complexity assumptions for several natural structural parameters including twin cover, modulator to stars, and modulator to paths numbers. Through these results, we delineate precise boundaries within a well-studied hierarchy of structural parameters, establishing where in the hierarchy meta-tractability results are possible.

Acknowledgements

Naomi and Amer, I am forever grateful to you. Your rigorous feedback and steady encouragement shaped both this dissertation and my development as a researcher. I feel truly fortunate to have had such welcoming and supportive advisors at my side.

Ryuhei Uehara, Chaitanya Swamy, Therese Biedl, and Richard Peng, thank you for your thoughtful questions and expertise that enriched my defense.

Collin, Jack, Kriti, and Sonja, you filled graduate school with laughter, friendship, and adventure. From study marathons to shared meals and weekend escapes, you made both school and life in Canada lighter and more memorable. I could not have asked for better friends to share this journey with.

Joseph, Pascale, and Christelle, your love is the ground from which my ambition took root. For the countless sacrifices and boundless faith that always carry me forward, my gratitude is endless and enduring.

Finally, to my husband, Nils: your unwavering love and constant support have been indispensable. You celebrated each small step of mine as if it were a triumph, and I look forward to many more journeys together. This dissertation, and every step I take in life, is brighter and stronger because of you.

Dedication

To my husband, Nils Hendrik Lukas and to my mother, Pascale Saliba.

Table of Contents

Examining Committee	ii
Author’s Declaration	iii
Abstract	iv
Acknowledgements	vi
Dedication	vii
List of Figures	xii
List of Tables	xxi
1 Introduction	1
2 Background	7
2.1 Background	7
2.2 Results Overview	26
3 Preliminaries	30
3.1 Foundational Graph Theory	30
3.2 Complexity Theory	37

3.3	Graph Problems	41
3.4	Solution Discovery Problems	45
3.5	Reconfiguration Problems	48
3.6	Structural Graph Parameters	50
3.7	Algorithmic Techniques	52
4	Multi-Sliding	54
4.1	Fixed-Parameter Tractability of UNLABELED DIRECTED/UNDIRECTED TOKEN MOVING for Parameter k	55
4.2	Fixed-Parameter Tractability of UNLABELED DIRECTED/UNDIRECTED TOKEN MOVING for Parameter b	58
4.3	W[1]-hardness of LABELED UNDIRECTED/DIRECTED TOKEN MOVING for parameter b	68
5	Solution Discovery for Problems in P	79
5.1	MATCHING-DISCOVERY	80
5.1.1	NP-hardness of MATCHING-DISCOVERY	81
5.1.2	Fixed-parameter tractability of MATCHING-DISCOVERY for Param- eter k	86
5.1.3	Polynomial Kernel for MATCHING-DISCOVERY with Parameter k	91
5.1.4	W[1]-hardness of MATCHING-DISCOVERY for Parameter b	93
5.2	VERTEX/EDGE CUT-DISCOVERY	106
5.2.1	Hardness of VERTEX/EDGE CUT-DISCOVERY	106
5.2.2	Fixed-Parameter Tractability of VERTEX/EDGE CUT-DISCOVERY for Parameter k	110
5.2.3	No Polynomial Kernel for VERTEX/EDGE CUT-DISCOVERY with Parameter k	114
5.3	SHORTEST PATH-DISCOVERY	124
5.3.1	NP-hardness of SHORTEST PATH-DISCOVERY	124

5.3.2	No Polynomial Kernel for SHORTEST PATH-DISCOVERY with Parameter $k + b$	128
5.3.3	Fixed-Parameter Tractability of SHORTEST PATH-DISCOVERY for Parameter k	132
5.3.4	Fixed-Parameter Tractability of SHORTEST PATH-DISCOVERY for Parameter b	134
6	Solution Discovery for NP-hard Problems	136
6.1	XNLP-hardness of INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY for Parameter pathwidth	137
6.1.1	The reduction framework	137
6.1.2	XNLP-hardness of INDEPENDENT SET-DISCOVERY for Parameter pathwidth	145
6.1.3	XNLP-hardness of VERTEX COVER-DISCOVERY for Parameter pathwidth	149
6.1.4	XNLP-hardness of DOMINATING SET-DISCOVERY for Parameter pathwidth	153
6.2	No Polynomial Kernel for INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY with Parameter $b + \text{pathwidth}$	157
6.2.1	The or-cross-composition framework	158
6.2.2	No Polynomial Kernel for INDEPENDENT SET-DISCOVERY with Parameter $b + \text{pathwidth}$	159
6.2.3	No Polynomial Kernel for VERTEX COVER-DISCOVERY with Parameter $b + \text{pathwidth}$	164
6.2.4	No Polynomial Kernel for DOMINATING SET-DISCOVERY with Parameter $b + \text{pathwidth}$	168
6.3	Polynomial Kernel for INDEPENDENT SET-DISCOVERY with Parameter k on every nowhere dense class of graphs	172
6.4	Polynomial Kernel for DOMINATING SET-DISCOVERY with Parameter k on every biclique-free class of graphs	176

7	Investigating Meta-Theorems for Solution Discovery	179
7.1	Fixed-Parameter Tractability of MSO-Discovery for Neighborhood Diversity	181
7.2	Hardness of FO-DISCOVERY for Modulator to Stars and Modulator to Paths Numbers	184
7.2.1	Hardness of FO-DISCOVERY for Modulator to Stars Number	190
7.2.2	Hardness of FO-DISCOVERY for Modulator to Paths Number	195
7.3	Hardness of FO-DISCOVERY for Twin Cover Number	197
7.4	Hardness of MSO-DISCOVERY for Bandwidth	204
8	Conclusion and Directions For Future Work	220
	References	224
	APPENDICES	244
A	Logic Formulas	245
A.1	Formulas for FO-DISCOVERY for Modulator to Stars and Modulator to Paths Numbers	245
A.1.1	Formulas for FO-DISCOVERY for Modulator to Stars Number	245
A.1.2	Formulas for FO-DISCOVERY for Modulator to Paths Number	246
A.2	Formulas for FO-DISCOVERY for Twin Cover Number	247
A.3	Formulas for MSO-DISCOVERY for Bandwidth	248

List of Figures

2.1	A subset of possible structural parameters. An arrow between two boxes indicates the existence of a function in the parameter above that lower-bounds the parameter below.	23
3.1	A simple graph G with five vertices ($V(G) = \{v_1, v_2, v_3, v_4, v_5\}$) and 6 edges ($E(G) = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_2, v_5\}, \{v_4, v_5\}\}$).	31
3.2	A simple directed graph D with five vertices ($V(D) = \{v_1, v_2, v_3, v_4, v_5\}$) and 6 arcs ($A(D) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4), (v_2, v_5), (v_4, v_5)\}$). The arcs represent ordered pairs indicating direction of the binary relation.	32
3.3	A vertex-colored graph where v_2 has multiple colors (blue and green), demonstrating that vertices can belong to multiple color classes. Here $\mathcal{C}(v_2) = \{C_1, C_2\}$ where C_1 is green and C_2 is blue.	33
3.4	Examples of a path and a cycle.	33
3.5	Examples of a tree, a forest, and a clique.	34
3.6	Induced subgraph and vertex deletion for $X = \{v_1, v_2, v_4\}$	35
3.7	A bipartite graph with partite sets $A = \{a_1, a_2, a_3\}$ (blue) and $B = \{b_1, b_2, b_3\}$ (green) and a non-bipartite graph with an odd-length cycle.	35
3.8	A 2-degenerate graph with degeneracy ordering shown using numbers. Each vertex has at most two adjacent vertices that appear “later” in the ordering.	36
3.9	A matching $M = \{v_1v_2, v_3v_4\}$ shown in (a) has saturated vertices v_1, v_2, v_3, v_4 while v_5, v_6 are unsaturated. The path $v_6v_3v_4v_5$ is an M -augmenting path. Using this augmenting path, we obtain the maximum matching $M' = \{v_1v_2, v_6v_3, v_4v_5\}$ shown in (b), which saturates all vertices.	42

3.10	Examples of vertex and edge cuts separating s from t . The s - t edge cut $\{e_3, e_4\}$ in (b) corresponds to the vertex cut $\{e_3, e_4\}$ in the line graph drawn in (c).	43
3.11	An independent set $I = \{v_1, v_3, v_5\}$ (shown in red). Note that no two vertices in I are adjacent, while the remaining vertices form a vertex cover.	44
3.12	A graph G and its tree decomposition of width $\max(B_i) - 1 = 2$	51
3.13	Additional structural graph parameters.	52
3.14	An example of a sunflower (with pink, blue and green petals) and a common core (the shaded intersection area). The black vertices represent elements that belong to individual petals but not to the core, while the green vertices represent the elements in the core that are shared by all three sets.	53
4.1	Formation of a segment in $\vec{M}'$; sources and target vertices for consecutive values of i are depicted by rows of circles, offset multi-slide by dashed green lines, matching multi-slides by solid black lines, and the linking multi-slide by a dashed blue line.	57
4.2	Sequences and $G_{\vec{M}'}$	60
4.3	Example circle graph with cycle segments highlighted, in-pools marked in brown, and out-pools marked in yellow, and sources and sinks of linking moves shown.	65
4.4	(a) K (b) G (c) K' showing level numbers.	70
4.5	(a) Supernodes of D with vertex nodes corresponding to a subgraph isomorphic to K' shown as unfilled circles. (b) Source, target, and storage gadgets for D_6 with edges shown using solid, dashed, and dotted arcs, respectively. Two tokens are initially present in the source gadget for D_6 (labeled d and e), and one node is in the target gadget and must receive the token with label a . (c) A schematic diagram showing supernodes as rectangles and source gadgets and target gadgets as labeled incoming and outgoing black arrows. For example, the source gadget attached to D_6 has two source gadget nodes whose tokens have target nodes in out-neighbors D_2 and D_3 , and the target gadget attached to D_6 has one node, the target node of a token in in-neighbor D_1	71

4.6	Edge-paths for superedge $D_{6,2}$ connected to Q_3 in the clock gadget (dashed lines represent long paths). Red nodes represent the left and right end of a roping path and the blue nodes represent the top and bottom ends of a storage path. Regions highlighted in yellow, green, brown, and blue show where tokens currently are. The first two clock segments have moved their tokens to target vertices, thus the clock is at position 3.	73
5.1	Reduction from RAINBOW MATCHING to MATCHING-DISCOVERY.	82
5.2	The reduced graph H . The red edges show the final matching configuration after all token movements. The filled and unfilled nodes show that the graph is bipartite. The vertices u_1 , u_3 , and u_5 form the partite set A	83
5.3	Sunflower structure in Theorem 49: The “sets” are edges (shown in green) that all share a common vertex that is the sunflower core. Each edge has one unique vertex v_i (the petals). The proof shows that if we have $2k + 1$ such edges, at least one can be used to replace e' in the matching.	92
5.4	Switch-gadget structure and token configurations.	94
5.5	Vertex-gadget for v formed by concatenating $\kappa - 1$ switch-gadgets and connected to the vertex decider path. Each switch-gadget corresponds to an index $j \neq i \in [\kappa]$. The vertex node is the leftmost node of the gadget and the vertex select node is the rightmost node. Red edges highlight the initial placement of tokens.	95
5.6	When all switch-gadgets are in the alternative configuration, an augmenting path (shown using the arrow) exists between the vertex decider node and the vertex select node.	96
5.7	Edge-gadget structure and token flow.	97
5.8	Overview of the connections between an edge-block for $\{1, 2\}$ and vertex-blocks for 1 and 2. Color class V_1 has vertices u_1 and u_2 and color class V_2 has vertices v_1 and v_2 . The graph G has edges u_1v_1 and u_2v_2 . Chains are shown as dashed lines for simplicity, with the destination edge of each chain connecting to the corresponding switch node.	99
5.9	Reduction from CLIQUE to VERTEX CUT DISCOVERY.	108

- 5.10 The instance selection mechanism via a binary tree structure. Eight instance components (shown as rectangles) have their local special s nodes form the leaves of a binary tree structure rooted at the global special s node. Three selector nodes, one at each depth of the tree, are each adjacent to all nodes at their respective depth level (green edges show these adjacencies). Each selector node can block one branch at its depth. With these $\log_2 8 = 3$ tokens making binary choices, exactly seven paths can be blocked, selecting precisely one instance. Each component's local special t node connects to the global special t node. Dashed lines represent paths of length $m^3 + \log t$. 116
- 5.11 Internal structure of component H^j showing two layers. Edge nodes are colored with the color of the edges they represent. Dashed or dotted edges are paths of length $m^3 + \log t$. Blue ones show some connections between nodes representing same-colored edges. Orange ones show some connections between nodes representing incident edges. Black ones illustrate the between-layer connections that force copying the selection made at the second copy nodes of the previous layer. Black dotted edges show the connectivity of special nodes. Reservoir edges for copy 2 in layer 2 connect to all second copy nodes in layer 2 of this instance component and all other instance components. Only one pool of reservoir edges is drawn for clarity but four exist. 118
- 5.12 The reduced graph H for a HAMILTONIAN PATH instance on graph G with n vertices. Green edges show how transition paths connect: if G contains edge v_1v_2 , then a transition path of three nodes connects to the select node for v_1 in layer i and the select node for v_2 in layer $i + 1$, and another transition path of three nodes connects to the select node for v_2 in layer i and the select node for v_1 in layer $i + 1$. Starting and ending paths also consist of three nodes each. Edge verifier paths have three nodes with tokens, where the middle node is adjacent to the middle nodes of all relevant paths. Red nodes initially have tokens. Dashed lines represent paths with $2n + 1$ internal nodes. 126

5.13	Or-cross-composition of two instances G_1 and G_2 . Red nodes have initial tokens. Green edges show how transition paths connect select nodes between consecutive layers: if G_1 contains edge v_1v_2 , then a transition path connects the select node $(1, 1, 1)$ to the select node $(1, 2, 2)$ and another transition path connects the select node $(1, 1, 2)$ to the select node $(1, 2, 1)$. Dashed lines represent edges that connect edge verifier paths to all relevant paths across both instances. Dash-dotted lines represent paths with $2n+1$ internal nodes.	131
6.1	An edge-gadget for edge $e = uv$ with weight $\omega(e)$. The constraint gadget ensures that the edge is oriented away from exactly one of its endpoints, while the $\omega(e)$ reservoir paths each provide one token to be distributed. The facilitator nodes (e, u) and (e, v) control token flow to the vertex-gadgets of u and v , respectively. Gray arrows show connections to both vertex-gadgets.	139
6.2	A vertex-gadget for vertex v . The central node connects to a control path and r receiving paths that can each receive at most one token. For each incident edge e with weight $\omega(e)$, there are $\omega(e)$ relay paths. Each relay path has a first checkpoint (receiving tokens from edge-gadgets) and a second checkpoint (adjacent to the central node, sending tokens to receiving paths). Gray arrows show incoming connections from edge-gadgets.	140
6.3	Sketch of the graph H constructed by the reduction for an MMO instance with graph G having vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = vw$ (weight 1), with $r = 3$. The supplier gadget with five donor paths ($rn - \Sigma = 5$) is shown at the bottom, with the supplier node adjacent to all backdoor nodes of the receiving paths (through orange edges).	141
6.4	The graph H for INDEPENDENT SET-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = vw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each edge-gadget has reservoir edges (with reservoir and pump nodes) and a constraint edge. Each vertex-gadget has a central node, control node, receiving nodes, and relay edges connecting from their checkpoint 1 and 2 nodes to the edge-gadgets through edges.	146

6.5	The graph H for VERTEX COVER-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each reservoir path has an auxiliary node, a pump node, and a reservoir node. Each receiving edge has one frontdoor node and one backdoor node. The supplier gadget with five donor paths ($rn - \Sigma = 5$) is adjacent to all backdoor nodes.	150
6.6	The graph H for DOMINATING SET-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each reservoir path has four nodes. Each relay path has a guard node between checkpoints. Communication paths have police nodes. Receiving paths have one frontdoor, one middle, and one backdoor node. The supplier gadget with five donor paths (each with seven nodes) is adjacent to all backdoor nodes. The edge added to H with a dominator node is omitted but connects to all checkpoint 2 nodes.	154
6.7	Part of the instance selector of the graph H^+ of INDEPENDENT SET-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. The orientations-hub node is adjacent to all constraint nodes (black dashed edges), the weights-hub node is adjacent to all pump nodes (green dashed edges), and the unselect node for j is adjacent to all the reservoir and central nodes in H^j (orange dotted edges).	161
6.8	Part of the instance selector of the graph H^+ of VERTEX COVER-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. Each hub node is adjacent to $b + 1$ pendant nodes. The orientations-hub node is adjacent to all facilitator nodes (black dashed edges), the weights-hub node is adjacent to all backdoor nodes of receiving edges (green dashed edges), and the select node for j is adjacent to all facilitator and checkpoint 2 nodes in H^j (orange dotted edges).	165

6.9	Part of the instance selector of the graph H^+ of DOMINATING SET-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. The orientations-hub node is adjacent to all facilitator nodes (black dashed edges), the weights-hub node is adjacent to all backdoor nodes (green dashed edges), and the select node for j is adjacent to all facilitator and guard nodes in H^j through paths of length two (orange dotted edges).	169
6.10	An example of a sunflower (with pink, blue and green petals) and a common core (the shaded intersection area). This sunflower is formed by the closed neighborhoods of the vertices in B_j of Theorem 119. This holds because no vertex from outside of X can be in the closed neighborhood of two vertices in B_j and the vertices of B_j have the same closed neighborhood in X (the green colored vertices).	175
7.1	A subset of possible structural parameters. An arrow between two boxes indicates the existence of a function in the parameter above that lower-bounds the parameter below.	180
7.2	A star vertex-gadget for vertex $v \in V_3$ with $\iota(v) = x$, for an instance with $\kappa = 4$, the vertex root nodes for $u, w \in V_3$, and the vertex block node b_3	187
7.3	A star edge-gadget representing $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$, the edge block nodes for uw and ab (edges between V_i and V_j), and the edge block node b_{ij} . The edge-gadget for uv is formed of $2n$ nodes, n corresponding to u and n corresponding to v	188
7.4	A path vertex-gadget for vertex $v \in V_3$ with $\iota(v) = x$, for an instance with $\kappa = 4$. The vertex block node b_3 is adjacent to all nodes of path vertex-gadgets for all vertices in V_3 , but it is omitted from the figure for clarity.	188
7.5	A path edge-gadget for edge $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$. It is formed of $2n$ nodes, n corresponding to u and n corresponding to v . The edge block node b_{ij} is adjacent to all nodes of path vertex-gadgets for all edges between V_i and V_j , but it is omitted from the figure for clarity.	189

7.6	Part of the colored graph $(H, \mathcal{C})$ formed by the reduction corresponding to the modulator to stars number. It shows the star edge-gadget of an edge $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$, and relevant parts of the two star vertex-gadgets for u and v . Additionally, the block nodes b_i , b_j , and $b_{i,j}$ are shown connecting to their respective root nodes. For u , the first x nodes of the group corresponding to V_j are adjacent to $\text{index}(i, \{i, j\})$ and the remaining $n - x$ nodes are adjacent to $\text{remainder}(i, \{i, j\})$. Similarly, for v , the first y nodes of the group corresponding to V_i are adjacent to $\text{index}(j, \{i, j\})$ and the remaining $n - y$ nodes are adjacent to $\text{remainder}(j, \{i, j\})$	190
7.7	Part of the colored graph $(H, \mathcal{C})$ formed by the reduction corresponding to the twin cover number. It shows the part of the twin cover and the cliques that correspond to two vertices $u, v \in V(D)$ and the edge $uv \in A(D)$ with $ L_{uv} = 3$. A line from a vertex or arc node to a clique means that the node is adjacent to all nodes of the clique. The nodes of the demand clique are yellow and those of the reservoir clique are brown. Supply nodes are orange, index nodes are teal, vertex nodes are purple, and arc nodes are black. (a) Since $ L_{uv} = 3$, we have three u -supply cliques for uv and three v -supply cliques for uv . (b) Each u -supply clique for uv corresponding to the i th supply pair contains $\Sigma^2 - \Sigma \cdot (i - 1)$ teal (C_5) index nodes and x_{uv}^i orange (C_4) supply nodes. Each v -supply clique for uv corresponding to the i th supply pair contains $\Sigma \cdot (i - 1)$ teal index nodes and y_{uv}^i orange supply nodes.	201
7.8	A sketch of the isolated and connected side rails for some vertex $u \in V(D)$ as discussed in Theorem 141. Demand nodes at the top of the connected side rail are colored yellow; reservoir nodes at the bottom of the isolated side rail are brown. Reservoir nodes contain tokens. The thin dotted lines indicate additional nodes that are omitted for clarity.	207
7.9	A sketch of the arc-gadget for some edge $uv \in A(D)$. Only the orange token-bearing nodes contain tokens. Here, the x and y values for the first supply tuple for uv are 2 and 4, respectively. The values for the supply tuple $t - 1$ for uv are 3 and 1, respectively, and those for the supply tuple t for uv are 1 and 2, respectively.	209

7.10	A sketch of how the connected side rail for some vertex $u \in V(D)$ and the connected side rail for some vertex $v \in V(D)$ connect to an arc-gadget for arc $uv \in A(D)$. Here v is assigned type d in the neighborhood of u , and u is assigned type c in the neighborhood of v . The thin dotted lines indicate additional nodes that are omitted for clarity.	210
------	---	-----

List of Tables

2.1	Summary of background and our results for the reconfiguration problems defined in this thesis; that is, UNLABELED UNDIRECTED TOKEN MOVING (UUTM), UNLABELED DIRECTED TOKEN MOVING (UDTM), LABELED UNDIRECTED TOKEN MOVING (LUTM), and LABELED DIRECTED TOKEN MOVING (LDTM), considering only unrestricted graphs. Here, n is the number of vertices, k is the number of tokens, and b is the budget. APX-hardness implies NP-hardness. “3-approx.” indicates the existence of a 3-approximation algorithm.	26
2.2	Summary of background and our results for solution discovery problems where the underlying property Π is a classical problem that is solvable in polynomial time; namely, matching, vertex/edge cut, shortest path, and spanning tree. Here, n is the number of vertices, k is the number of tokens, and b is the budget. All background results are for unrestricted graphs, and we only specify graph classes for our results when they differ and are relevant for comparison.	28

Chapter 1

Introduction

Change is a fundamental characteristic of real-world systems. From automated warehouses that adapt to shifting demand patterns to quantum circuits that reallocate qubits for optimal computation, real-world systems must continuously reconfigure themselves. Traditional algorithmic approaches, designed for static environments where problems are solved once from scratch, fail to capture this complexity of the real world that often requires the *transformation* of one *configuration* of one system into some other through a sequence of intermediate states. This gap between static solutions and dynamic reality motivates this thesis.

Consider the challenge faced by automated parking systems in dense urban environments [170]. Vehicles are arranged on sliding platforms within a grid structure, where each car can only move by sliding along rails to empty adjacent positions. When retrieving some vehicles from deep within the structure, the system cannot simply lift them out. Instead, it must carefully orchestrate a sequence of platform movements, sliding cars along predetermined car-free paths to create paths to the exits. This physical constraint, where transformations must proceed through local, edge-based movements, appears across diverse domains: mobile robots navigating warehouse floors can only move along guide paths, modules storing items in dense three-dimensional grids of modern warehouses can only slide to adjacent positions [77], and trains in a depot can only travel along connected tracks.

The quantum computing domain also exemplifies these edge-based movement transformations through its fundamental architectural constraints. Modern quantum processors feature sparse connectivity graphs in which physical qubits can only interact with their immediate neighbors [64, 130]. During compilation, logical qubits from a quantum program

must be mapped onto these physical qubits, but this initial mapping rarely suffices for the entire computation [169]. As the algorithm executes and requires two-qubit operations between previously non-adjacent logical qubits, the compiler must dynamically reconfigure the mapping through *swap* operations. Each swap slides quantum information between adjacent physical qubits, gradually moving logical qubits into positions where they can interact. Another example of such transformations arises from urban bike-sharing systems. In such systems, bicycles must be redistributed every few hours to match pre-specified required numbers at each station for coming demand. Rebalancing trucks collect bikes from stations where commuters accumulated them and must discover a distribution ensuring that every station meets its minimum required inventory.

Across all these examples, minimizing the cost of transformation is not merely an optimization goal, but a critical operational constraint. In automated parking systems, each unnecessary platform movement extends retrieval time, frustrating customers and reducing throughput. For warehouse modules, excessive travel leads to delivery delays. In quantum computing, every additional swap operation extends circuit time and introduces errors that can destroy delicate quantum computation. For bike-sharing systems, minimizing truck travel distance directly impacts operational profitability and the system’s ability to meet demands. Whether measured in time, energy, computational accuracy, or monetary cost, the efficiency of most system transformations determines the viability of the entire system.

These real-world constraints naturally also lead to a graph-based model to study transforming configurations. In this abstraction, vertices represent valid positions, tokens on vertices represent the entities being moved (cars, logical qubits, bikes), while edges encode allowed transitions between positions. For bike-sharing systems, the graph is simply a map of the cities, and trucks can travel along routes. For quantum circuits, vertices represent physical qubits, edges connect those qubits that can directly interact, and tokens represent logical qubits that can only swap between connected positions (that is, tokens sliding along an edge). In a similar way, we can model diverse physical systems challenges into a unified framework: the study of how tokens slide along graph edges to transform one configuration into another.

We let these real-world systems challenges inspire our work, and thus focus our research efforts on:

Understanding and developing efficient algorithms for token edge-based movement problems on graphs, where we seek to transform configurations through minimum size sequences of such token edge-based movements.

The choice to focus on token edge-based movement is not arbitrary; it directly reflects the physical and logical constraints of many real systems, as we saw. The edge-based

movement model encompasses more than one fundamental movement paradigm. In some systems, each individual edge traversal incurs a unit cost; every time a parking platform slides one position, a quantum swap occurs, or a warehouse robot moves between adjacent waypoints, we consume a resource. In some others, we pay the unit cost per token movement along a token-free path regardless of the length of the path traversed by the token. Consider having a collection of atoms each randomly assigned to one distinct square of a large flat grid. In order to reassign atoms (to different squares) so that they form a condensed arrangement, e.g., all squares in the center of the grid, atoms can be moved one by one along a trajectory of empty squares. Minimizing the number of atoms to move is one of the challenges faced by physicists in preparing given configurations of atoms for use in quantum simulators [165, 52, 50, 76].

The definition of what constitutes one *step* in the configuration transformation process depends on the underlying cost model. When each individual edge traversal incurs a unit cost, one step consists of a *token slide*; that is, moving a token from one vertex to an adjacent vertex along a single edge. When we pay unit cost per token movement regardless of path length, one step consists of a *token multi-slide*; moving a token along an entire path of multiple edges (provided the path is free of other tokens). In both models, applying one such step transforms the current configuration into the next configuration in the transformation sequence of transformation steps. Both mentioned edge-based movement models prohibit teleportation or discontinuous jumps, enforcing the locality constraints that make these problems realistic and computationally challenging.

Note also that in these transformation scenarios, the nature of the *target configuration* of the system varies fundamentally. Sometimes we know precisely which target configuration to *reconfigure* our current configuration to. For example, in the example of preparing atoms for use in quantum simulators, we know that all atoms must end up in a specific predetermined arrangement in the center of the grid. We call these *reconfiguration* problems [150], where both the *initial* and target configurations are fully specified. However, at other times, we must discover a suitable *final configuration* as part of the solution process. In bike-sharing redistribution, we need any distribution where each station meets its minimum inventory; many final configurations could work. We call these *solution discovery* problems [83], where we seek any final configuration that satisfies certain properties while minimizing the transformation cost from the current infeasible state.

The transition from asking “does a valid configuration exist?” to “how can we efficiently transform between configurations?” marks a fundamental and recent shift in algorithmic thinking toward modeling that change which is an unavoidable aspect of our environment. To understand these transformation problems with unspecified final configurations, we first study how they behave when the final configuration must satisfy properties defined

by classical graph subset problems. These are problems whose solutions consist of selecting subsets of vertices or edges from a graph, such as choosing which physical qubits to map logical ones to, or which grid squares to occupy with atoms, or which set of road intersections must be occupied to monitor all roads in a city network. These selections can be easily modeled as tokens placed on vertices or edges, with at most one token per element. Graph subset problems provide a rich foundation on which to build and have widespread applications in organ exchange networks [162], pandemic contact tracing [177], sensor placement optimization [186], social influence maximization [161], and quantum computing, as demonstrated in our examples.

Beyond these fundamental graph subset problems, we extend our study to encompass final graph configurations (again, with at most one token on an element) that must satisfy some logical formula given as part of the problem input (that is, not fixed). These logical formulas are flexible rules that can be used to encode expressive requirements that capture the constraints of many real-world systems [122, 73, 140]. For example, this broader study naturally captures scenarios like bike-sharing systems, where valid configurations must satisfy a non-standard graph subset problem requirement: a minimum number of bikes at each station. By moving from specific graph subset problems to general logical formulas, we can model virtually any system where configurations must satisfy a collection of rules. This generalization allows one to develop unified algorithmic techniques and deeper theoretical insights that apply across diverse domains, rather than creating specialized solutions for each application.

Under reconfiguration problems, we study the impact of both labeled and unlabeled tokens to model the effect of distinguishable versus indistinguishable items. Labeled tokens represent distinguishable entities where identity matters: specific cars must reach designated exits in parking systems. In contrast, unlabeled tokens model indistinguishable items where only the distribution pattern matters: atoms need only form the correct geometric arrangement regardless of which atom occupies which position. This distinction will fundamentally affect the problems' complexities as seen in previous work [56] and as we will see. While labeled token problems must track individual identities, unlabeled versions enjoy additional flexibility that can lead to shorter transformation sequences and different algorithmic approaches.

Finally, we mention that most of the problems we will study are [56] or will turn out to be classically computationally hard. Thus, we also study these problems under the parameterized complexity paradigm [69]. *Parameterized complexity* is a branch of computational complexity theory that is different from classical complexity theory and examines the complexity of algorithmic problems not only in relation to the input size (i.e. n), but also with respect to one or more parameters of the problem instances. While

classical complexity theory might consider a problem *intractable* (or computationally hard) due to exponential worst-case running time with respect to the input size, parameterized complexity recognizes that real-world problems’ instances often have special structures that can be exploited. By identifying relevant parameters; such as the number of tokens to move, the maximum degree of the input graph, or the size of the transformation sequence we seek, we can develop algorithms whose running time is exponential only in these typically small parameters in real life while remaining polynomial in the input size. This means that instances with thousands of vertices but small parameter values can be solved efficiently (or are *tractable*), transforming seemingly intractable problems into ones with practical algorithms for many real-world applications.

Our transformation problems are amenable to such complexity analysis, as they are defined over graphs having a bounded maximum degree, or a bounded diameter, and or the problems themselves could be seeking a bounded size transformation or having a bounded number of tokens. All of those bounds are examples of parameters that one can consider when designing parameterized algorithms for such problems. When we succeed in designing a (*fixed-parameter tractable*) algorithm that runs in some time with respect to the parameter, but in polynomial time with respect to the size of the instance (that is, n^c for a constant c), we say that the problem is *fixed-parameter tractable* when parameterized by k . When we cannot find such algorithms, we prove that the problem is hard for complexity classes such as $W[1]$, $W[2]$, or even $XNLP$, providing evidence that a fixed-parameter tractable algorithm for that parameterization is unlikely [69]. These hardness results guide us toward identifying which parameters offer genuine algorithmic advantages and which do not.

This thesis will address the fundamental need to understand and design efficient algorithms for token edge-based reconfiguration and solution discovery problems on graphs. This calls for investigating both the computational complexity landscape of these problems and the development of practical algorithms that can handle real-world instances. We specifically work towards that research goal through this summary of objectives:

1. Study the complexity of solution discovery problems with unlabeled tokens, whose final configuration must satisfy properties defined by classical graph subset problems (with at most one token per element) and where a step consists of a token slide.
2. Study the complexity of solution discovery problems with unlabeled tokens, whose final configuration must satisfy some graph property expressed as a logical formula given as part of the problem input (with at most one token per element) and where a step consists of a token slide.

3. Study the complexity of reconfiguration problems with labeled and unlabeled tokens and where a step consists of a token multi-slide.

Notice that with unlabeled tokens, reconfiguration problems where a step consists of a token slide reduce to a matching problem. Given our objectives, from here on, we associate solution discovery problems with a step that consists of a token slide and reconfiguration problems with a step that consists of a token multi-slide. In the following section, we present the necessary background, including the preliminaries, previous, and related work.

Chapter 2

Background

Having outlined our research objectives, we now provide the necessary foundation for our technical contributions. The following chapter surveys the relevant previous and related work, motivates the questions we answer in the thesis, and introduces along the way some algorithmic techniques that we employ.

2.1 Background

In this thesis, we study solution discovery problems where the fundamental transformation step is the token slide. Given a graph G with a vertex set $V(G)$ and an edge set $E(G)$, an *initial configuration* $S \subseteq V(G)$ (or $S \subseteq E(G)$) of size k (represented as tokens placed on the elements of the set S), and a *budget* b , a *solution discovery problem* asks whether we can transform S into a *final configuration* $T \subseteq V(G)$ (or $T \subseteq E(G)$) of size k that satisfies a fixed graph property Π using at most b token slides. A *token slide* consists of moving a single token from its current vertex (edge, respectively) to an adjacent vertex (edge, respectively). We also study *reconfiguration problems* with labeled and unlabeled tokens and where the fundamental transformation step is the token multi-slide. In these problems, we are given a graph G with a vertex set $V(G)$ and an edge set $E(G)$, *initial* and *target configurations* S and T of size k representing labeled or unlabeled tokens placed on distinct vertices of G , and a *budget* b , and the goal is to answer whether we can transform S into T using at most b multi-slides. A token *multi-slide* consists of moving a single token from its current vertex to another vertex via a path of vertices containing no other tokens. The two described edge-based movement constraints (with no jumps to non-adjacent positions, only movement

along the edges (vertices, respectively) of the graph) reflect the physical limitations of many real-world systems, where entities can only transition through connected states.

In what follows, we start by introducing the *solution discovery* and *reconfiguration* frameworks under which our problems fall. We survey initial results showing that our reconfiguration problems are *NP-hard* (that is, classically computationally hard) and survey/present further results later in the thesis that prove that even properties solvable in polynomial-time can yield NP-hard solution discovery problems. We conclude from this that we can address this intractability through the lens of parameterized complexity theory. *Parameterized complexity theory* analyzes a problem’s complexity through both its input size n (for graph problems, n is the number of vertices) and problem-specific parameters (in our case, the number of tokens k , the budget b , and some structural graph parameters), distinguishing fixed-parameter tractable problems from those that are not, that is, are $W[1]$ -hard or worse.

Fixed-parameter tractable problems are those for which we can design a (*fixed-parameter tractable*) *algorithm* that on any input of size n with parameter value k , runs in some time with respect to k but polynomial time with respect to n (that is, in n^c time for some constant c). This means that the possibly exponential or worse complexity is entirely confined to the parameter, making such algorithms practical when the parameter value is small in real-world instances, even if the input size is large. Note that problems with polynomial-time (with respect to n) algorithms (that is, efficiently solvable problems) are also fixed-parameter tractable with any parameter. We provide background on parameterized complexity and briefly present some key techniques used under this paradigm, including color coding for finding small substructures and derandomization for converting randomized algorithms into deterministic ones. We make use of these techniques in some of the algorithms we present, and thus include a description of them in the background.

Another important technique in parameterized complexity is *kernelization*, which transforms preprocessing from an ad hoc practice into a rigorous mathematical framework. We say that two instances are *equivalent* if they are both yes-instances or both no-instances of the same problem. A kernelization or *kernel* is a fixed-parameter tractable algorithm that reduces any instance of a parameterized problem into an equivalent one whose size and parameter are bounded above solely in terms of the original instance parameter. When this function is polynomial, we have a *polynomial kernel*, which represents the gold standard for preprocessing effectiveness. Unfortunately, not all problems admit polynomial kernels; we study whether our solution discovery problems possess this desirable property with respect to certain parameters, and thus present a background section on this technique, which we discuss in more technical detail in the preliminaries.

We then provide a background section on descriptive complexity. Many of our results on solution discovery problems will be ones for which the property Π is defined by a classical graph problem. For example, when Π is defined by the classical matching problem which asks whether there exists a set of k edges with no two edges sharing a common vertex, the corresponding Π -DISCOVERY (in this case, MATCHING-DISCOVERY) problem asks whether we can transform an initial configuration of k tokens on k edges into any valid matching of size k through at most b token slides. Descriptive complexity, through logical expressiveness, will help us move from analyzing the complexity of individual solution discovery problems to speaking of the tractability of all solution discovery problems where Π can be expressed in a certain type of logic. This approach yields powerful meta-theorems: a single result about logically-definable problems can simultaneously capture the tractability of hundreds of specific problems, eliminating the need for repetitive case-by-case proofs and revealing deep structural patterns across problem families. In this context, we study the parameterized complexity of our solution discovery problems with popular structural graph parameters.

Finally, we briefly survey other related problems and frameworks to clarify how our solution discovery and reconfiguration problems differ from other approaches. We give an overview of our detailed results after the background, but throughout include questions that arise from the concepts presented and that we will work on answering in the thesis. For the sake of the background, a *graph class* is a (typically infinite) set of graphs that share a common structural property. We define relevant graph classes later in the preliminaries and only those needed for comprehension of the proofs and importance of the results will be defined. When the focus is on the containment relation between classes and not on their definitions, we mention the relation. The same applies for problems or structural graph parameters.

We begin with the reconfiguration framework before examining subsequent concepts.

The Reconfiguration Framework

In the reconfiguration framework [123, 175, 150], different *types* of reconfiguration framework problems are defined over classical ones such as graph and satisfiability problems. In particular, given a problem (equivalently, property) Π , a *reconfiguration framework problem* takes as input an instance I and asks an algorithmic or structural question defined over the *reconfiguration graph* where the vertices represent solutions to Π on instance I and two solutions are adjacent when one can be transformed into the other with one transformation step. Here, a transformation step modifies the current solution in a prescribed way (such as sliding a token to an adjacent vertex under token sliding when Π is a graph

subset problem with solutions viewed as tokens on vertices or edges of the graph), and a transformation sequence is a walk in this reconfiguration graph. Crucially, unlike solution discovery problems, every intermediate configuration in the transformation must remain feasible (equivalently, is a solution) with respect to Π . The framework is general enough to accommodate colored and uncolored tokens, labeled and unlabeled tokens, as well as any conceivable transformation step, including but not limited to *(multiple) token sliding*, that is, sliding (multiple) tokens to adjacent vertices or edges, *(multiple) token jumping* [135], that is, moving (multiple) tokens to arbitrary vertices or edges, *token addition/removal*, that is, changing the number of tokens, and other transformation steps.

Reconfiguration framework problems of algorithmic flavor typically take additionally as input two feasible solutions S and T to Π on instance I . The problem is then one of the following incomplete list of types: a *reachability problem*, that is, does there exist a sequence of transformation steps that converts S into T while maintaining feasibility throughout; a *shortest transformation problem*, that is, what is the minimum number of transformation steps needed to convert S to T ; a *bounded reachability problem*, can S be transformed to T using at most b transformation steps. Structural framework problem types include but are not limited to, a *connectivity problem*, that is, can any feasible solution be transformed to any other; a *diameter problem*, what is the maximum size of a minimum size transformation sequence between any two feasible solutions, and other problems that ask structural questions such as “What are necessary and sufficient conditions for the input so that its reconfiguration graph is a collection of trees” [11], “Does the reconfiguration graph have a Hamiltonian path (that visits every vertex once) regardless of the instance” [3], and others. When Π is defined by a classical problem, such as matching, we call the corresponding Π -RECONFIGURATION framework problem, MATCHING-RECONFIGURATION, and include the type in the problem name when necessary to distinguish between variants, such as MATCHING-REACHABILITY-RECONFIGURATION or MATCHING-SHORTEST TRANSFORMATION-RECONFIGURATION.

Recently, another type of reconfiguration framework problem has been investigated: the *optimization variant*, where we start from a feasible but non-optimal solution (such as a non-maximum independent set) and seek to reach an optimal solution (such as a maximum independent set) while maintaining feasibility throughout [124]. Unlike the previously mentioned algorithmic types that require both initial and target configurations, this variant takes only a single solution and asks for a more desirable solution reachable from it. Ito et al. introduced this type and studied INDEPENDENT SET-OPTIMIZATION-RECONFIGURATION [124] then Blanché et al. studied DOMINATING SET-OPTIMIZATION-RECONFIGURATION [25]. Both of these problems were shown to be PSPACE-hard (generally believed to be harder than NP-hard) even on very restricted graph classes. It should be

noted that solution discovery problems remain distinct from optimization variants under the reconfiguration framework, which require all intermediate configurations to remain feasible solutions (such as feasible independent sets), thus inheriting the computational hardness of reconfiguration framework decision problems, which are often harder than solution discovery problems (that is, they are often PSPACE-hard).

The reconfiguration framework includes our reconfiguration problem; however, note that here the problem or property upon which this problem is defined is a trivial “does a set of k vertices exist in the graph?”. Călinescu, Dumitrescu, and Pach proved that the reconfiguration problem with unlabeled tokens is not only NP-hard but also APX-hard [49]. If a problem is APX-hard, then one cannot efficiently find solutions arbitrarily close to optimal, though constant-factor approximations may still be possible. On the positive side, the authors provided an approximation algorithm (with constant factor 3), and showed that for tree graph classes, the problem can be solved optimally in linear time. They also established that at most k multi-slides are always sufficient (and sometimes necessary) for reconfiguring k unlabeled tokens in any connected graph. For labeled tokens, the problem is also NP-hard and APX-hard and $7k/4$ multi-slides are always enough, and $3k/2$ are sometimes necessary, for the reconfiguration of k labeled tokens in the infinite planar rectangular grid graph.

Following this, this question emerges as the logical progression for our transformation problem.

Question 1. What is the parameterized complexity of the reconfiguration problems with labeled or unlabeled tokens and parameters such as the number of tokens k or the budget b ?

Within the reconfiguration framework, Mouawad et al. [147] were the first to consider the parameterized complexity of reconfiguration framework problems. After that, a myriad of classical graph problems have been studied under the reconfiguration framework for their bounded transformation or other variants. We survey here related results that emerged from studying bounded transformation variants (with token sliding as a transformation step) of classical graph problems, which also serve as the defining properties Π for the solution discovery problems we focus on in this thesis.

In particular, INDEPENDENT SET RECONFIGURATION is PSPACE-hard even on bounded treewidth graphs [17] (where treewidth is a structural graph parameter that measures how “tree-like” a graph is), polynomial-time (with respect to n) solvable on certain graph classes related to trees [181, 113] and others [125]. With respect to the number of tokens k , INDEPENDENT SET RECONFIGURATION is W[1]-hard on the class of split graphs [15], the

class of bipartite graphs and other graph classes [13] and in FPT on the class of bipartite graphs without 4-cycles [13], the class of bounded degree graphs, the class of planar graphs and others [15, 17, 14]. For DOMINATING SET RECONFIGURATION, Bousquet et al. [38] showed that the problem is PSPACE-hard even on graphs of treewidth at most 12. However, the problem admits polynomial-time algorithms for the class of tree graphs and the class of interval graphs [134]. In their survey, Bousquet et al. [40] showed that the problem is fixed-parameter tractable when parameterized by b on some graph classes that include every nowhere dense class of graphs. For classical graph problems with efficient polynomial-time algorithms, Bousquet et al. [39] showed that SHORTEST PATH RECONFIGURATION is W[1]-hard when parameterized by $k + b$ but fixed-parameter tractable when parameterized by b on every nowhere dense class of graphs and when parameterized by other structural parameters such as treedepth, cluster deletion number, or the modular width.

We note that the reconfiguration variant of the independent set problem is often also referred to by using different names such as TOKEN SLIDING [15], or SLIDING TOKEN [61].

The Solution Discovery Framework

Our solution discovery problems fall under the broader solution discovery framework introduced by Fellows et al. [83], which also encompasses various transformation steps beyond token sliding. Like its reconfiguration counterpart, the general framework allows non-graph problems and any conceivable transformation step, including but not limited to token sliding, token jumping, token addition/removal, and other modification rules, with the first three being among the most studied modification rules relating to graph subset problems in the solution discovery and reconfiguration frameworks. However, as Fellows et al. [83] and we demonstrate, the choice of what is considered a transformation step fundamentally affects the computational complexity and theoretical richness of solution discovery problems: while MATCHING-DISCOVERY remains polynomial-time solvable under token jumping and token addition/removal, it becomes NP-hard under token sliding. This sensitivity to the movement choice motivates our focus on the most restrictive yet practically relevant operation, token sliding along edges.

The solution discovery framework itself emerged from the recognition that many optimization scenarios involve repairing or adapting existing configurations rather than computing solutions from scratch. In the introductory paper by Fellows et al. [83], the authors focus on problems where Π represents a graph property that is defined by a classical graph problem such as matching, shortest path, vertex cover, independent set, or dominating set, and the solution discovery problem would then seek any final configuration that is a

feasible solution to the problem Π reachable from an initial configuration within the transformation budget. This is different from reconfiguration framework problems that require the maintaining of feasibility with respect to Π at every step of the transformation. The relaxation of feasibility at every intermediate step of the transformation makes solution discovery problems computationally easier than their reconfiguration framework (decision) counterparts and allows different algorithmic approaches.

The solution discovery framework generalizes earlier work by Garnero et al. [94] on the *proper coloring* discovery problem. Here, a *coloring* of a graph is an assignment of exactly one color from a fixed color set to each vertex, and a *proper coloring* is one that additionally assigns distinct colors to adjacent vertices. If the coloring does not do so, then we call it *improper*. In their proper coloring discovery problem, Garnero et al. [94] focus on finding an arbitrary proper coloring of a given graph G , starting from a given improper coloring, by recoloring the minimum number of vertices. Similarly, the framework also generalizes the earlier work of De Biasi and Lauri [21], who studied the same problem with a different transformation step that allows the swapping of colors on two adjacent vertices. The framework also encompasses the MOBILE FACILITY LOCATION problem introduced by Demaine et al. [62] in which existing facilities must be relocated (using token slides) and clients assigned to minimize the (weighted) sum of movement and service costs. This problem and its variants have seen significant algorithmic theoretical (approximation algorithms and inapproximability results) and heuristic progress in the uncapacitated [89, 108, 4, 9], capacitated [160], and even the online setting [98, 97]. Demaine et al. [63] also developed a general framework for minimizing token slides to achieve one of many allowed final configurations. Other problems falling under the framework include the s - t -CUT-SUM problem studied by Bilò et al. [23] for which the authors give an inapproximability result and in which the goal is to move (using token slides) the tokens to form a set of vertices whose removal disconnects to given vertices s and t . The introductory paper of Fellows et al. [83] can be thought of as unifying these approaches under a common theoretical umbrella.

Garnero et al. [94] show that their proper coloring discovery problem (where the problem Π asks whether there exists a proper coloring for the input graph) is NP-hard whenever we have a fixed number of three or more colors. Additionally, it is W[1]-hard when parameterized by the budget, but fixed-parameter tractable when parameterized by the budget and the number of colors. In contrast, De Biasi and Lauri [21] show that their proper coloring discovery problem (with the swapping of colors on adjacent vertices being a transformation step) is W[1]-hard when parameterized by the budget for every fixed number of colors. Demaine et al. [62] proved NP-hardness for the problem of minimizing the number of token slides needed so that the final configuration is a path between two fixed vertices,

provided an approximation algorithm for the same problem (with constant factor cn for some constant c), and later [63] proved that when minimizing the number of token slides needed in order to get the tokens into a final configuration whose vertices induce a graph from a fixed group of graphs, then the parameterized complexity with respect to the number of tokens of any problem of a subset of such problems depends on the treewidth of the edge-deletion minimal graphs in the group. The latter implies that when minimizing the number of token slides to get the tokens into a final configuration whose vertices form a connected graph or a path between two fixed vertices, then the problem is fixed-parameter tractable with respect to the number of tokens, but when the vertices of the final configuration are non-adjacent, form a 2-connected graph, or a grid, then the same problem is $W[1]$ -hard. In the following unifying work of Fellows et al. [83], the focus expanded to a comprehensive study of several (other) classical and fundamental graph problems, particularly vertex cover, independent set, dominating set, and coloring. We note that the results of Demaine et al. [63] cannot speak of properties defined by problems such as dominating set, vertex cover, and $(s-t)$ vertex cut, where the feasibility of the final configuration additionally depends on the input graph.

While it is obvious that all the solution discovery problems of Fellows et al. [83] would inherit their NP -hardness (equivalently, their classical computational complexity) from the NP -hardness of the property or equivalently the classical problem Π , the behavior of these problems under parameterized complexity assumptions is not as obvious. VERTEX COVER-DISCOVERY was shown to be NP -hard, fixed-parameter tractable when parameterized by the number of tokens k , $W[1]$ -hard when parameterized by the budget b , and in XP for parameter treewidth [83]. XP is the class of parameterized problems solvable by an algorithm that runs in some time with respect to the parameter k and polynomial time with respect to n , but the degree of that polynomial may depend on the parameter. In contrast, INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY were shown to be NP -hard and $W[1]$ -hard for parameters b and $k + b$ on all graphs [83].

The problems become slightly less computationally complex when we restrict the type of graph in the input [83]. For example, on every nowhere dense class of graphs, all three problems (VERTEX COVER-DISCOVERY, INDEPENDENT SET-DISCOVERY, and DOMINATING SET-DISCOVERY) become fixed-parameter tractable when parameterized by the budget b . For parameter k , INDEPENDENT SET-DISCOVERY becomes fixed-parameter tractable on the class of d -degenerate graphs and every nowhere dense class of graphs, and DOMINATING SET-DISCOVERY becomes fixed-parameter tractable on every class of biclique-free or semi-ladder-free graphs. The class of d -degenerate graphs and every nowhere dense class of graphs are standard *sparsity* benchmarks. A graph class is *sparse* if its graphs have few edges relative to vertices (typically, $|E| = O(|V|)$), and in modern usage, if dense

(that is, non-sparse) substructures, primarily cliques, are avoided in its graphs regardless of the radius: even if one replaces disjoint groups of vertices lying within distance at most this radius by single nodes and then examines the connections those nodes inherit, large cliques still do not appear. Biclique-free, semi-ladder-free graph classes, while not necessarily sparse in the above sense, contain d -degenerate graphs and every nowhere dense class of graphs and generalize many of the algorithmically favorable properties of sparse graphs. Together, nowhere dense, d -degenerate, biclique-free, semi-ladder-free graph classes subsume many natural sparse families of graphs, such as the class of planar graphs, which contains all graphs that can be drawn on a plane without any edges crossing; edges may be drawn as continuous curves connecting their endpoints.

We add to the list of studied solution discovery problems where Π is defined by a classical and fundamental graph problem with efficient polynomial-time algorithms the following. We single out *matching* among properties in P because it probes a different axis of structure: it enforces *disjointness* (no two chosen edges share a vertex), in contrast to *cut* (separation) and *shortest path* (connectivity/metric). It also comes with an exceptionally mature theory that we can leverage for both algorithmic upper bounds as well as lower-bound constructions.

Question 2. What is the classical complexity of MATCHING-DISCOVERY?

As demonstrated above and as we will see later, solution discovery problems, even those where Π is defined by a classical graph problem with polynomial-time algorithms, can turn out to be classically computationally hard (NP-hard). This computational intractability necessitates a more refined analysis of these problems, which can be done through the lens of parameterized complexity theory [69, 58, 85]. This framework can help identify tractable cases of the problems by isolating problem-specific parameters that may be small in practice and for which efficient algorithms exist.

Question 3. What is the parameterized complexity for each of MATCHING-DISCOVERY, VERTEX/EDGE CUT-DISCOVERY, and SHORTEST PATH-DISCOVERY for parameters such as the number of tokens k or the budget b ?

Parameterized Complexity and Tools

Parameterized complexity, pioneered by Downey and Fellows [68], has emerged as a crucial theoretical framework but extends far beyond that, offering practical solutions to computationally hard problems in diverse domains. The framework has been used in various

areas of computer science, such as database theory, artificial intelligence, and computational biology. In bioinformatics, parameterized algorithms have proven essential for its NP-hard problems including sequence assembly and analysis as well as genome comparison and completion [46], and researchers like Hüffner et al. [120] provide comprehensive surveys of these applications. The impact extends to machine learning, where the parameterized complexity analysis of training ReLU networks has revealed fundamental insights about the influence of data dimensionality on computational complexity [90], and recent work has applied parameterized complexity analysis to explain decisions in ML models, addressing critical challenges in explainable AI [152]. In computational social choice, the framework has become indispensable for analyzing problems in voting procedures, preference aggregation in multiagent systems, bribery, and control in elections [44, 43].

As mentioned earlier, the core idea of parameterized complexity is to measure the running time of algorithms not just in terms of the input size n , but also with respect to a parameter k that captures some structural or other aspect of the problem instance [69]. When a problem is fixed-parameter tractable when parameterized by some parameter, we say it is in the class *FPT*. For instance, Fellows et al. [83] showed that VERTEX COVER-DISCOVERY when parameterized by the number of tokens k is in *FPT*, with an algorithm running in time $c \cdot 2^k \cdot n^3$ for a constant c .

The weaker notion of tractability in parameterized complexity is captured by the class *XP*. For problems in this class, the exponent of the polynomial in the running time may depend on the parameter, which can result in impractically large running times even for small parameter values. Fellows et al. [83] showed that the vertex subset discovery problems they study fall into this class when parameterized by treewidth. Indeed, treewidth has become one of the most important and well-studied parameters in parameterized complexity, with a recent survey describing it as “the most well studied width measure in the field” [146] with researchers in diverse areas investigating how their problems behave when parameterized by this structural measure: Courcelle’s celebrated theorem [55] shows that any graph property expressible in a certain logic can be decided in linear time (with respect to n) on graphs of bounded treewidth; in database theory, query evaluation becomes fixed-parameter tractable for various query languages when parameterized by the treewidth of the query structure [84, 102]; and in constraint satisfaction, fundamental results show that such problems can be solved in polynomial time on instances of bounded treewidth [142, 107]. Given the importance of this measure, we ask the following.

Question 4. Are INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY fixed-parameter tractable when parameterized by treewidth?

Unfortunately, not all parameterized problems admit fixed-parameter tractable algorithms. The *W-hierarchy* provides a framework for classifying parameterized problems that are believed to be inherently difficult (that is, not admit fixed-parameter tractable algorithms) from a parameterized perspective. This hierarchy consists of complexity classes $\text{FPT} = \text{W}[0] \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \text{W}[3] \subseteq \dots$, where inclusions are conjectured to be strict [69]. That is, it is formed of complexity classes $\text{W}[i]$ for every integer i , such that $\text{W}[i] \subseteq \text{W}[i+1]$, and it is a subset of XP . A problem that is *W[1]-hard* is believed to not admit a fixed-parameter tractable algorithm, under the widely accepted assumption that $\text{FPT} \neq \text{W}[1]$. For solution discovery problems, Fellows et al. [83] demonstrated several $\text{W}[1]$ -hardness results: for example, both INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY are $\text{W}[1]$ -hard for parameters b and $k + b$. Beyond the W -hierarchy, problems can be shown to be *XNLP-hard*, which implies hardness for every level of the W -hierarchy and suggests that even space-efficient (that is, polynomial space) parameterized algorithms are unlikely [34]. As we will see, several solution discovery problems (that is, those appearing in Question 4) exhibit this stronger form of hardness when parameterized by structural parameters like treewidth and pathwidth (which measures how “path-like” a graph is and is, for every graph, larger than treewidth).

The development of parameterized algorithms has been derived by several fundamental techniques that have proven successful across diverse problem domains. Among the most influential are bounded search trees, dynamic programming on tree decompositions, iterative compression, bidimensionality theory, color coding, derandomization and kernelization [58].

Color Coding. One influential technique is the powerful *color coding* randomized technique introduced by Alon, Yuster, and Zwick [7] that revolutionized the search for small substructures in graphs. The core idea of this technique is elegantly simple: to find a k -element substructure in a graph, randomly assign colors from $\{1, \dots, k\}$ to vertices (or edges) and search for a “colorful” substructure where each element has one distinct color. While a specific k -element solution might not be colorful in a random coloring, it becomes colorful with probability at least $k!/k^k \geq e^{-k}$ when we use k colors. By repeating the process $e^k \cdot k$ times, we find the k -element substructure with high probability. The technique transforms the problem of finding a size- k structure into finding a colorful one, which is often significantly easier; finding a size- k path is hard, but finding a colorful path using dynamic programming on color subsets takes only $2^k \cdot n^c$ time for some constant c . From its original applications to paths and cycles [7], color coding has expanded to subgraph isomorphism [141], network motif detection in biological networks [5], temporal graph exploration [80], and the detection of signaling pathways in protein-protein interaction (PPI)

networks [121]. Recent advances include structure-aware coloring strategies [154] that exploit graph properties to reduce dynamic programming calls by orders of magnitude in kidney exchange and transportation optimization problems.

Derandomization Using Perfect Hash Families. While color coding provides efficient randomized algorithms, many applications require deterministic guarantees, achieved through *derandomization* using *perfect hash families*. An (n, k, k) -*perfect hash family* is a collection $\mathcal{F}$ of functions from $\{1, \dots, n\}$ to $\{1, \dots, k\}$ such that for every k -element subset $S \subseteq \{1, \dots, n\}$, there exists a function $f \in \mathcal{F}$ that is injective on S . Instead of random colorings, we systematically try each function from a perfect hash family, guaranteeing that any k -elements substructure will be “colorful” under at least one function. Naor, Schulman, and Srinivasan [149] provided the foundational construction that achieves families of size $e^k k^{c \log k} \log n$ (for some constant c) that can be generated in polynomial time, with subsequent advancements by Alon and Gutner [6]. This derandomization technique is invaluable beyond color coding, for example, for communication complexity, where perfect-hashing-based two-round protocols for list problems (originating with Orlitsky) remain the canonical construction and are explicitly referenced in recent treatments [45, 153].

Kernelization. Perhaps no parameterized complexity technique has had a wider impact than kernelization, a mathematical framework that elevated preprocessing from informal methods to principled science. Michael Fellows dubbed it the “lost continent of polynomial time” [82], recognizing that while practitioners had long known preprocessing can dramatically improve performance, kernelization provides the first theoretical lens to analyze these ubiquitous data reduction steps with mathematical precision. This framework has transformed practice well beyond theory: modern SAT solvers use kernelization-inspired variable elimination techniques to handle industrial instances with millions of variables [75], reducing them by orders of magnitude. In computational biology, kernelization principles guide the design of feature selection algorithms for genomic data, where reducing from tens of thousands of genes to the most informative subset is crucial for both computational feasibility and biological interpretability [178, 163].

While every fixed-parameter tractable problem admits some kernelization algorithm [48], the existence of a polynomial kernel, which is particularly desirable from a practical point of view, is not guaranteed. Bodlaender et al. [31, 35] developed the first framework for proving kernel lower bounds, providing the first rigorous impossibility results for preprocessing and explaining why some problems resist decades of reduction efforts. Fortnow and Santhanam [88] showed a connection to the complexity hypothesis $\text{NP} \not\subseteq \text{coNP}/\text{P}$. Specifi-

cally, for several problems, a polynomial kernel with respect to a parameter would imply that $\text{NP} \subseteq \text{coNP}/\text{P}$, and thus an unlikely collapse of the polynomial hierarchy to its third level [184].

Given that, for parameter k , Fellows et al. [83] show that INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY are fixed-parameter tractable on every nowhere dense class of graphs and every class of biclique-free graphs, respectively, we ask whether they also admit polynomial kernels.

Question 5. Does INDEPENDENT SET-DISCOVERY parameterized by k admit a polynomial kernel on every nowhere dense class of graphs? Does DOMINATING SET-DISCOVERY parameterized by k admit a polynomial kernel on every biclique-free class of graphs?

Question 6. Do any of MATCHING-DISCOVERY, VERTEX/EDGE CUT-DISCOVERY, and SHORTEST PATH-DISCOVERY when parameterized by k or b admit polynomial kernels?

Descriptive Complexity

Descriptive complexity, which has proven effective in addressing complexity-theoretic questions for static problems [55], offers a promising direction for understanding solution discovery problems by moving away from case-by-case analysis where each problem’s unique structure demands its own specialized techniques and proofs. While problem-specific analysis can yield deep insight into individual problems, it can leave us without a unifying framework to understand broader patterns across problem families. *Descriptive complexity* is a powerful framework that classifies problems not by the computational resources they require (time or space), but by the logical expressiveness needed to define them [122]. This shift in perspective, from operational to declarative, has revolutionized how we approach algorithmic problems across diverse domains.

The power of this logical approach is exemplified by Courcelle’s celebrated theorem [53]. This single meta-theorem has captured the tractability of hundreds of NP-hard problems, from Hamiltonian cycle to graph coloring, when restricted to graphs of bounded treewidth, eliminating the need for problem-specific algorithms. The theorem’s impact extends far beyond classical graph problems: Burton and Downey [47] extended it to simplicial complexes for computing quantum invariants of 3-manifolds, and Grohe [105] used it to show that the problem of computing crossing numbers is fixed-parameter tractable. Even in database theory, recent work has shown how logic on appropriate graph representations produces efficient algorithms for consistent query answering [145].

In this thesis, logic will be defined on (colored) graphs. Here, a *colored graph* is a tuple $(G, \mathcal{C})$ where $\mathcal{C}$ is a set of vertex sets (colors), and a vertex in G can belong to zero or multiple vertex sets in $\mathcal{C}$ (or equivalently, have no or multiple colors). Logic at its core allows us to make statements such as “There exists a vertex with color red that is connected to a vertex with color blue” or “There exists a set of vertices forming a clique of size κ ”. The ability to express these and even more complex structural properties or problems in a formal language is what enables researchers to handle many different problems within a single theorem or algorithm. In both *first-order logic* (FO) and *monadic second-order logic* (MSO), *formulas* are constructed over (colored) graphs. Each formula constructed over (colored) graphs is a sentence that describes a property of a graph. This sentence can be true or false for any given graph. The key difference between the aforementioned logics lies in what they allow: FO allows one to say “there exists a vertex” while MSO additionally allows one to say “there exists a set of vertices”.

In both logics, a formula uses *vertex variables* (like x and y) which act as placeholders for vertices in the graph, similar to how in algebra x represents an unknown number. There also are *set variables* (like X) which represent collections of vertices. A set variable X is a *free set variable* when it has not been specified which set it represents; it is left open so that one can later ask questions like “Which sets make this formula true?”. Unlike in Courcelle’s theorem (and other theorems for non-transformation problems), since we would like to describe properties of the final configuration and not the graph, we work with a variant of FO that includes one free set variable X , to which the formula can refer (but not quantify over). The basic statements available to build a formula are: $X(x)$ meaning that the vertex x belongs to the set X ; $C_i(x)$ meaning that the vertex x has color C_i ; $E(x, y)$ meaning that there is an edge between the vertices x and y ; and $x = y$ meaning that the vertices x and y are the same. These basic statements are combined using logical connectives ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) and quantification over vertices with “ $\exists x \in X :$ ” (“there exists a vertex $x \in X$ such that”) and “ $\forall x \in X :$ ” (“every $x \in X$ satisfies”). For example, $\exists x \in V(G) : C_1(x) \wedge \exists y \in V(G) : (C_2(y) \wedge E(x, y))$ says “there exists a vertex with color C_1 adjacent to some vertex with color C_2 ”.

Monadic second-order logic (MSO) extends (our formulation of) FO by allowing set variables beyond the free set variable and quantification over all sets. In MSO, it is possible to have multiple set variables (like Y , Z) and to quantify over them: “ $\exists Y \subseteq X :$ ” means “there exists a set $Y \subseteq X$ such that” and “ $\forall Y \subseteq X$ ” means “every set $Y \subseteq X$ satisfies”. For a formula $\phi(X)$ with a free set variable X and for a set $S \subseteq V(G)$, we write $G \models \phi(S)$ when ϕ is true for the graph G with X interpreted as vertex set S . Checking whether $G \models \phi(S)$ can be done in time polynomial in n and the size of the formula [74]. MODEL CHECKING asks, given a graph G and a formula $\phi(X)$, if there exists a set S such

that $G \models \phi(S)$. We will refer to FO and MSO with no free set variable as *classical* FO and MSO, respectively. For more background on logic, we refer to the book by Wilfrid Hodges [115]. We note that the number of colors used in a formula ϕ is always bounded by its size, thus if the size of the formula given as part of the input is treated as a constant (or parameter), as will be the case in this thesis, one can assume that there is a constant number of colors $|\mathcal{C}|$ in the colored graph that is part of the same input.

The descriptive complexity approach offers several key advantages. First, it provides a uniform treatment of a group of problems. For instance, Courcelle’s theorem states that classical MSO MODEL CHECKING can be decided in linear time on graphs of bounded treewidth and extends to graphs with bounded cliquewidth [54], which is another structural parameter smaller than treewidth. A similar theorem states that classical FO MODEL CHECKING can be solved in almost linear time on every nowhere dense class of graphs [106] and in polynomial time on every monadically stable class of graphs (which are broader classes that extend nowhere dense ones while preserving their key algorithmic or logical properties) [71, 70]. Second, it reveals deep connections between seemingly unrelated problems that share the same logical complexity and enables the transfer of algorithmic techniques across domains. For instance, Kolman et al. [131] showed that classical MSO-definable optimization problems on graphs of bounded treewidth admit compact extended formulations, bridging descriptive complexity with linear programming theory. Kneis et al. [128] developed game-theoretic algorithms for Courcelle’s theorem that avoid the state explosion problem of automata-theoretic approaches, making Courcelle’s theorem applicable to larger instances in practice. For data management under uncertainty, Amarilli et al. [8] demonstrated that classical MSO on tree-like instances yields linear-time provenance circuits and tractable probabilistic evaluation, while Kazana and Segoufin [127] achieved constant delay enumeration for classical MSO queries on trees. Furthermore, Ganian et al. [92] show that problems expressible in classical MSO admit polynomial kernels on graphs with structural parameters like bounded rank-width.

Descriptive Complexity and the Reconfiguration Framework. Descriptive complexity has also proven to be valuable for transformation problems, such as those defined under the reconfiguration framework. Mouawad et al. [148] observed that Courcelle’s theorem immediately yields a meta-theorem for bounded transformation reconfiguration framework problems: if a classical problem’s feasible solutions are MSO-definable, then these reconfiguration variants (both with token sliding and token jumping) are fixed-parameter tractable when parameterized by $b + tw$ (where tw is the input graph’s treewidth). This extends to graphs with bounded cliquewidth, yielding tractability for the parameter $b + cw$ (where cw is the input graph’s cliquewidth). Similarly, these reconfiguration variants for

FO-definable problems are fixed-parameter tractable when parameterized by b on all graph classes admitting efficient FO MODEL CHECKING [40]. A similar observation holds for the solution discovery framework: if a problem is MSO-definable, then its solution discovery variants with the token jumping or token sliding transformation steps are fixed-parameter tractable when parameterized by $b + cw$, and when a problem is FO-definable, then its solution discovery variants are fixed-parameter tractable parameterized by b on every class that admits efficient FO MODEL CHECKING [168]. These results follow from the fact that formulas can be constructed to quantify transformation sequences whose length depends only on b , combined with the general MODEL CHECKING results for MSO and FO, respectively.

For reconfiguration framework problems, Wrochna [180] demonstrated that without the parameter b , several reconfiguration framework reachability variants of FO-definable problems become PSPACE-hard even on graphs with constant bandwidth (which is another structural parameter larger than both pathwidth and treewidth), highlighting a critical role of the parameter b . Despite this hardness, Gima et al. [99] explored meta-theorems for bounded transformation reconfiguration framework problems with the token jumping transformation step, an input containing an MSO-defined property, and without involving b , necessarily then requiring structural parameters larger than bandwidth or orthogonal to it. Parameters they considered are *modulator*-type parameters such as *minimum vertex cover*, *twin cover*, and *cluster deletion* numbers (but also includes *modulator to paths* and *modulator to stars* numbers) that typically isolate a small “difficult core” and leave a tractable remainder; and “type compression” measures like *neighborhood diversity* that bound the number of distinct neighborhoods, typically enabling compact state summaries. The authors showed that these problems are fixed-parameter tractable when the parameters are only neighborhood diversity and vertex cover.

Since similar observations hold for solution discovery variants as for the bounded transformation reconfiguration variants of FO- and MSO-definable problems when the parameter includes the budget b , we follow Gima et al.’s approach and investigate what happens when the parameter excludes b . Specifically, we study solution discovery problems whose transformation step is token sliding, whose input contains an MSO- or FO-definable property and that are parameterized by the same structural parameters of Gima et al., asking whether the relaxation of maintaining feasibility in intermediate configurations yields improved meta-tractability results for these parameters. Particularly, if FO-DISCOVERY denotes the solution discovery variants of FO-definable problems coupled with the token sliding transformation step and MSO-DISCOVERY denotes the solution discovery variants of MSO-definable problems coupled with the token sliding transformation step, we ask:

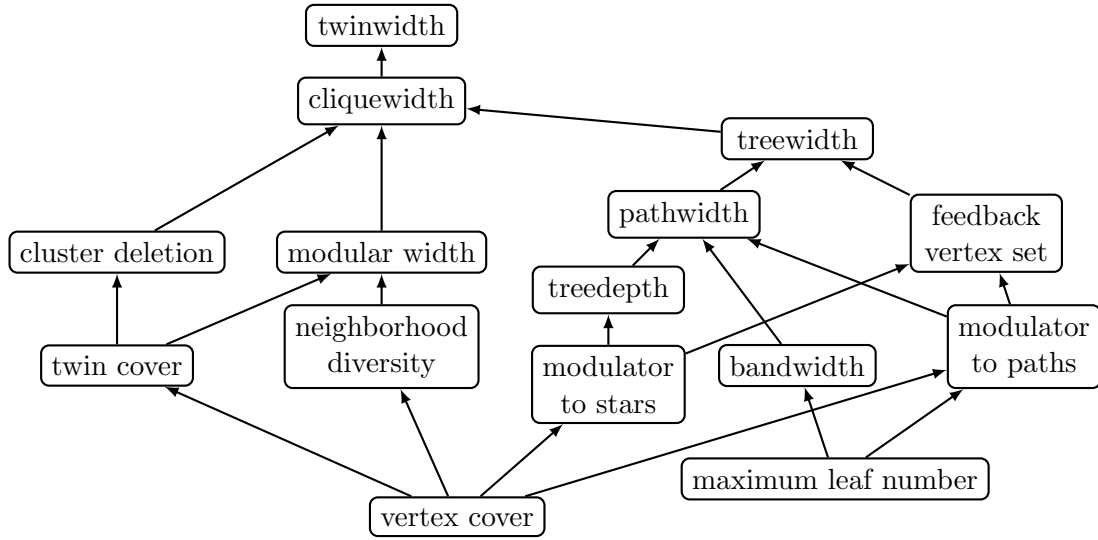


Figure 2.1: A subset of possible structural parameters. An arrow between two boxes indicates the existence of a function in the parameter above that lower-bounds the parameter below.

Question 7. Are FO-DISCOVERY and MSO-DISCOVERY fixed-parameter tractable when parameterized by any one of the parameters of Figure 2.1?

The dominance and incomparability relations charted in Figure 2.1 provide a roadmap for Gimma et al.’s and other structural parameters (the figure itself represents only a subset of known structural parameters). By locating algorithmic and hardness results within the hierarchy of this figure, it is possible to (i) identify the minimal parameter that restores tractability for FO- and MSO-DISCOVERY, and (ii) transfer results across parameters whenever the figure’s relations permit (positive results with respect to a certain parameter propagates downward resulting in positive results for the larger parameters and negative results propagate upwards resulting in negative results for the smaller parameters), yielding a tight frontier for where meta-tractability is possible and where barriers to meta-tractability start to arise.

Related Work

The reconfiguration framework encompasses various other *motion planning problems* with distinct movement rules and constraints, many having both discrete (that is, graph-based)

and continuous (that is, on the plane) variants. In PEBBLE MOTION or MULTIAGENT PATH FINDING (discrete), many agents move in parallel in synchronous transformation steps (from one vertex to an adjacent one) while ensuring no vertex/edge collisions [132, 185]; the continuous analogue of multiagent path finding (MULTI-ROBOT MOTION PLANNING) models robots, often unit disks, moving in the plane while avoiding collisions [118]. Recent work on these problems includes improved complexity bounds for trees in PEBBLE MOTION [27], and an anytime LNS-based solver that improves large-scale MULTIAGENT PATH FINDING solution quality [157]. For the continuous analogue of the latter, Abrahamson et al. [2] prove PSPACE-hardness for reconfiguring unit squares in simple polygons and unit disks in polygons with holes.

The reconfiguration framework also includes the TOKEN SWAPPING problem, where tokens placed on vertices of a graph need to be moved to their destination vertices by swapping tokens on adjacent vertices (in a manner akin to the requirement of swapping logical qubits between physical ones to ensure connectivity between those that need to interact). The main contribution of Bonnet et al. [37] is showing that TOKEN SWAPPING is $W[1]$ -hard when parameterized by the number of allowed swaps k . The authors also introduced SUBSET TOKEN SWAPPING as a generalization where a function specifies for each labeled token a set of possible destination vertices, rather than just one fixed destination. The goal is to reach any configuration where each token is placed on one of its allowed destinations within k swaps. This generalizes both TOKEN SWAPPING and COLORED TOKEN SWAPPING (where tokens and vertices have colors, and tokens must reach vertices of matching colors) which was proven to be NP-hard on trees [24]. The authors prove that even this most general variant is fixed-parameter tractable on nowhere-dense graph classes (including planar graphs and bounded treewidth graphs) when the parameter is the allowed number of token swaps. All the aforementioned variants under the reconfiguration framework highlight its richness; different movement rules, from discrete single-edge steps to continuous motion, from non-collision preservation to property maintenance, yield fundamentally different computational problems.

We note that other reconfiguration tasks are sought after for modular robotic systems; a recent advance gives reconfiguration algorithms for cubic modular robots under realistic movement constraints, with universality guarantees via plane-sweep and for shapes with sufficient external feature size [171]. Two recent surveys provide a broader context and taxonomies of strategies and mechanisms for these systems [59, 139].

A framework particularly close to solution discovery in the token sliding setting is concerned with minimizing movement of tokens into a final configuration that satisfies a predefined property. The framework was introduced by Demaine et al. [62] who then focused on final configurations satisfying the following properties: connectivity, indepen-

dence, and perfect matchability, while minimizing the number, the sum, or the maximum of movements. Their movement minimization problems demand that all given tokens induce a subgraph that satisfies the property, which means allowing multiple tokens to occupy a vertex in the final configuration. Nevertheless, they proved that minimizing the number of token slides needed to reach a final configuration (where each vertex contains one token) that is a path between two vertices is classically hard. While the aforementioned paper and others [19, 23] focused on approximation and inapproximability results, Demaine et al. [63] proved that with respect to the parameter number of tokens, the complexity of a subset of these graph problems (where the property is defined through the graph induced by the vertices of the final configuration belonging to a group of graphs) depends on the treewidth of the edge-deletion minimal graphs in the group. Other results also in the discrete setting provide approximation and inapproximation results for these problems (with properties concerned with radius and distance) under explicit per token distance or energy constraints [16]. Notice that some properties such as “tokens need to be on vertices whose removal disconnects two given vertices s and t ” do not need to move multiple tokens into a single vertex, thus becoming analogous to our solution discovery problems. We mention that Demaine et al. [63] achieved a similar treewidth-based characterization in the restrictive setting (as in the unrestrictive one) where each vertex hosts at most one token (as noted earlier). This framework has also been analyzed in the geometric domain. One specific example in the continuous setting is the problem SPREADING POINTS, where points must be separated by a minimum distance while minimizing movement, which has been studied both classically [138, 96] and through parameterized complexity [86]. In the plane, Tekdas et al. [78, 173] considered the problem of finding the final destinations of a subset of tokens that will form an s - t -path (bridge) that minimizes the number of tokens between s and t and the distance traveled by the tokens.

Beyond the aforementioned frameworks, several established “change” paradigms capture related algorithmic questions. In *dynamic graph algorithms*, the goal is to efficiently update information about graph properties, such as connectivity or spanning trees, while the graph undergoes updates such as edge insertions or deletions. Instead of recomputing the property from scratch after every update (which could take polynomial time), dynamic graph algorithms maintain auxiliary data structures that can be updated more efficiently, often in logarithmic or polylogarithmic time per change. This efficiency has been achieved for many fundamental problems, including maximal matchings [20], minimum spanning forests [174], and vertex cuts [116]. Under the *reoptimization framework*, a previously solved instance undergoes a small modification, and algorithms exploit the known solution to compute a solution to the modified instance more efficiently than solving it from scratch [22, 26]. A complementary approach is *local search via k -exchange neighborhoods*

(conceptually akin to multiple token-jumping in the optimization variant of the reconfiguration framework) where one iteratively improves a solution within a bounded-radius neighborhood; recent advances span new parameterized local search results for MAX- c -CUT [95], and improved k -median local search with a $(2.836 + \varepsilon)$ guarantee [51]. These frameworks formalize different notions of “change” (updates to the instance vs. constrained moves in solution space) but none is concerned with maintaining feasibility during the transformation between solutions or the “closeness” of solutions.

2.2 Results Overview

In Chapter 4, we work towards Question 1 and map the parameterized complexity landscape of our transformation problems with the token multi-sliding transformation step and pinpoint where tractability emerges. We do so particularly for the parameter b , that is, the budget. We prove a clean dichotomy with respect to token type: with unlabeled tokens the problem is fixed-parameter tractable when parameterized by either the number of tokens k or the budget b on both directed and undirected graphs, while with labeled tokens it is $W[1]$ -hard when parameterized by b . Table 2.1 summarizes the relevant background and our results.

Table 2.1: Summary of background and our results for the reconfiguration problems defined in this thesis; that is, UNLABELED UNDIRECTED TOKEN MOVING (UUTM), UNLABELED DIRECTED TOKEN MOVING (UDTM), LABELED UNDIRECTED TOKEN MOVING (LUTM), and LABELED DIRECTED TOKEN MOVING (LDTM), considering only unrestricted graphs. Here, n is the number of vertices, k is the number of tokens, and b is the budget. APX-hardness implies NP-hardness. “3-approx.” indicates the existence of a 3-approximation algorithm.

Problem	Parameterized complexity	Other results
UUTM	In FPT when parameterized by k or by b	APX-hard; 3-approx. [49]
UDTM	In FPT when parameterized by k or by b	APX-hard [49]
LUTM	$W[1]$ -hard when parameterized by b	APX-hard [49]
LDTM	$W[1]$ -hard when parameterized by b	APX-hard [49]

For our solution discovery problems, we begin with properties corresponding to classical problems that are solvable efficiently by polynomial-time algorithms. Table 2.2 summarizes the relevant background and our results on these problems. We focus on the following

properties: matching, vertex/edge cut, and shortest path. These primitives are canonical and ubiquitous, so they form a clean testbed for all complexity analysis. In Chapter 5, we answer Question 2 and prove exactly this: the discovery variant MATCHING-DISCOVERY is NP-hard. We strengthen the hardness result of Demaine et al. [62] and show using the same reduction framework that SHORTEST PATH-DISCOVERY is NP-hard even when the input graph is restricted to one from a more restrictive group of graphs that is sparse. This and the other NP-hardness results for solution discovery problems [62, 23] suggest a parameterized analysis focused first on the most fundamental parameters, the number of tokens k and the budget b . We continue Chapter 5 mirroring the introductory paper of Fellows et al. [83], that is, we chart the boundaries: all three problems are fixed-parameter tractable when parameterized by k (independently from the characterization result of Demaine et al. [63], we present a short proof of the fixed-parameter tractability of SHORTEST PATH-DISCOVERY with respect to parameter k here); SHORTEST PATH-DISCOVERY is also fixed-parameter tractable when parameterized by b ; and MATCHING-DISCOVERY and VERTEX/EDGE CUT-DISCOVERY are W[1]-hard with respect to b . These results advance progress on Question 3 and clarify, for parameters k and b , where parameterized tractability emerges and where locality constraints imposed by sliding remain a barrier, even for polynomially solvable properties.

Building on the results of Fellows et al. [83], who obtained only an XP upper bound for their vertex-subset discovery problems when parameterized by treewidth, we strengthen this result in Chapter 6. We prove that INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY are XNLP-hard when parameterized by pathwidth or by treewidth, even when a width-realizing path/tree decomposition is given. Consequently, these problems are W[i]-hard for every integer i , so fixed-parameter tractable algorithms are unlikely. This answers Question 4 in the negative. In Chapters 5 and 6, we also analyze kernelization exactly for those problems where fixed-parameter tractable algorithms are obtained, to quantify the limits of preprocessing. Thus, we show that MATCHING-DISCOVERY admits a polynomial kernel when parameterized by k , but VERTEX/EDGE CUT-DISCOVERY does not unless an unlikely assumption in complexity theory (namely, that $\text{NP} \subseteq \text{coNP/P}$) holds. The same lower bound applies for SHORTEST PATH-DISCOVERY when parameterized by $k+b$. For INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY, we show that the problems admit polynomial kernels with parameter k on every nowhere dense class of graphs and every biclique-free class of graphs, respectively. These results address Questions 5 and 6. For INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY, we additionally prove that no polynomial kernels exist when parameterized by $b+\text{pathwidth}$, even when a width-realizing path decomposition is given. The choice of $b+\text{pathwidth}$ is deliberate: b alone is

Table 2.2: Summary of background and our results for solution discovery problems where the underlying property Π is a classical problem that is solvable in polynomial time; namely, matching, vertex/edge cut, shortest path, and spanning tree. Here, n is the number of vertices, k is the number of tokens, and b is the budget. All background results are for unrestricted graphs, and we only specify graph classes for our results when they differ and are relevant for comparison.

Π	Our results	Previous Work
Matching	NP-hard; in FPT when parameterized by k ; W[1]-hard when parameterized by b	
Vertex/Edge cut	In FPT when parameterized by k ; W[1]-hard when parameterized by b	no $cn^{1-\epsilon}$ -approx. for any constant c and any $\epsilon > 0$ [23]
Shortest path	NP-hard even on a class of graphs that is sparse; in FPT when parameterized by k or by b	NP-hard; cn -approx. for constant c [62]; in FPT for parameter k [63]
Spanning tree		Poly-time algorithm [103]

too weak (all these discovery problems are W[1]-hard for b), but leads to fixed-parameter tractable algorithms on nowhere dense graph classes that include bounded pathwidth graph classes. This combination is a step in analyzing which minimal structural parameter can be combined with b to lead to a polynomial kernel.

Finally, we work toward answering Question 7 in Chapter 7. In particular, we show that MSO-DISCOVERY is fixed-parameter tractable when parameterized by neighborhood diversity, and thus alternatively by minimum vertex cover. Unfortunately, we obtain several hardness results; FO-DISCOVERY is W[1]-hard when the parameter is twin cover, modulator to stars, or modulator to paths number. This implies that there exists at least one property definable in FO for which the solution discovery problem is W[1]-hard when parameterized by any of those parameters. MSO-DISCOVERY is also W[1]-hard when parameterized by bandwidth. Recall from the background that negative results for a parameter imply negative results for smaller (higher level) parameter. That means that the negative result for twin cover directly implies the same negative result for cliquewidth, modular width, and twinwidth. To completely answer the question, it suffices to examine whether FO-DISCOVERY is fixed-parameter tractable when parameterized by bandwidth and maximum leaf number, we leave these as open problems.

Concluding remarks, open problems, and directions for future research are discussed in Chapter 8. All the results presented in this thesis have appeared in part or in full in one of the following conference papers or manuscripts.

- *Parameterized Complexity of Reconfiguration of Atoms*. In *WALCOM: Algorithms and Computation 2022*, LNCS 13174, pp. 263–274. Joint work with Alexandre Cooper, Amer E. Mouawad, and Naomi Nishimura.
- *Solution Discovery via Reconfiguration for Problems in P*. In *ICALP 2024*, LIPIcs 297, 76:1–76:20. Joint work with Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, and Sebastian Siebertz.
- *Kernelization Complexity of Solution Discovery Problems*. In *ISAAC 2024*, LIPIcs 322, 36:1–36:17. Joint work with Mario Grobler, Amer E. Mouawad, Naomi Nishimura, Vijayaragunathan Ramamoorthi, and Sebastian Siebertz.
- *Surveying Graph Parameters for Algorithmic Meta-Theorems for Solution Discovery*. Submitted, 2025. Joint work with Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura and Sebastian Siebertz.

Chapter 3

Preliminaries

This chapter formally defines the problems we study and presents the mathematical, graph-theoretic, and complexity-theoretic preliminaries required to ensure that diverse readers can understand our results. For comprehensive treatments of graph theory and complexity theory, we refer the reader to the excellent books by Douglas B. West [179], Christos H. Papadimitriou [155], Rodney G. Downey and Michael R. Fellows [69], and Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad [42]. We note that some section-specific preliminaries appear instead in sections on relevant results.

3.1 Foundational Graph Theory

Basic Definitions and Notation

We begin with the fundamental objects of our study. Throughout this thesis, we use $\mathbb{N} = \{0, 1, 2, \dots\}$ to denote the non-negative integers, $\mathbb{N}_{>0} = \{1, 2, \dots\}$ to denote the positive integers, and $\mathbb{R}_{\geq 0}$ to denote non-negative reals. For $k \in \mathbb{N}$, we define $[k] = \{1, 2, \dots, k\}$ with the convention that $[0] = \emptyset$. For any finite set S , we denote by $|S|$ its cardinality (the number of elements it contains). For any finite string x , we denote by $|x|$ its size (the number of symbols it contains).

A (*simple*) *graph* G is a representation of a binary symmetric relation. It is a pair $(V(G), E(G))$ where $V(G)$ is a finite set of *vertices* (also called *nodes*) and $E(G)$ is a set of *edges*. Each edge connects two vertices and can be represented as an unordered pair $\{u, v\}$ where $u, v \in V(G)$ and $u \neq v$. The vertices u and v are called the *endpoints* of the

edge $\{u, v\}$. With slight abuse of notation, we often write uv to denote the edge $\{u, v\}$ when clear from context. We let $|V(G)| = n$ and $|E(G)| = m$ represent the numbers of vertices and edges, respectively. See Figure 3.1. For a vertex $v \in V(G)$, its *neighborhood* $N(v) = \{u \in V(G) : uv \in E(G)\}$ consists of all vertices *adjacent* to v (that is, connected via an edge with v). We say that a vertex $v \in V(G)$ is *incident* to an edge $e \in E(G)$ (and vice versa) when v is one of the endpoints of e . The *closed neighborhood* is $N[v] = N(v) \cup \{v\}$, and the *degree* of v is $\deg(v) = |N(v)|$.

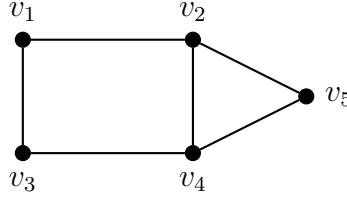


Figure 3.1: A simple graph G with five vertices ($V(G) = \{v_1, v_2, v_3, v_4, v_5\}$) and 6 edges ($E(G) = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_2, v_5\}, \{v_4, v_5\}\}$).

A (*simple*) *directed graph* (or (*simple*) *digraph*) D is a representation of a binary relation. It is a pair $(V(D), A(D))$ where $V(D)$ is a finite set of vertices and $A(D)$ is a set of arcs. Each arc points from one vertex to another and is represented by an ordered pair (u, v) where $u, v \in V(D)$. We use $A(D)$ to refer to the set of arcs to distinguish it from that of edges. Also, the vertices u and v are called the *endpoints* of the arc (u, v) . With slight abuse of notation, we often write uv to denote the arc (u, v) when clear from context. We let $|V(D)| = n$ and $|A(D)| = m$ represent the numbers of vertices and arcs, respectively. See Figure 3.2. We say that a vertex $v \in V(D)$ is *incident* to an arc $a \in A(D)$ (and vice versa) when v is one of the endpoints of a . For a vertex $v \in V(D)$, its *in-neighborhood* $N^-(v) = \{u \in V(D) : (u, v) \in A(D)\}$ consists of all *in-neighbors* of v (that is, all vertices that point to v via an arc). For a vertex $v \in V(D)$, its *out-neighborhood* $N^+(v) = \{u \in V(D) : (v, u) \in A(D)\}$ consists of all *out-neighbors* of v (that is, all vertices that v point to via arcs). For a vertex $v \in V(D)$, $N(v) = N^-(v) \cup N^+(v)$ and $N[v] = N(v) \cup \{v\}$, $N^-[v] = N^-(v) \cup \{v\}$, and $N^+[v] = N^+(v) \cup \{v\}$ refer to the *neighborhood*, *closed neighborhood*, *closed in-neighborhood*, and *closed out-neighborhood* of v , respectively. The *in-degree* of v is $\deg^-(v) = |N^-(v)|$, the *out-degree* of v is $\deg^+(v) = |N^+(v)|$, and the *degree* of v is $\deg(v) = |N(v)|$.

These local neighborhood concepts extend to sets: for $S \subseteq V(G)$, we define $N(S) = \bigcup_{v \in S} N(v) \setminus S$ and $N[S] = N(S) \cup S$. For $S \subseteq V(D)$, we define $N^-(S) = \bigcup_{v \in S} N^-(v) \setminus S$ and $N^-[S] = N^-(S) \cup S$, $N^+(S) = \bigcup_{v \in S} N^+(v) \setminus S$ and $N^+[S] = N^+(S) \cup S$, $N(S) = \bigcup_{v \in S} N(v) \setminus S$ and $N[S] = N(S) \cup S$.

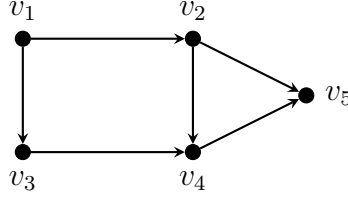


Figure 3.2: A simple directed graph D with five vertices ($V(D) = \{v_1, v_2, v_3, v_4, v_5\}$) and 6 arcs ($A(D) = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4), (v_2, v_5), (v_4, v_5)\}$). The arcs represent ordered pairs indicating direction of the binary relation.

A *multigraph* is a generalization of a graph that allows multiple edges between the same pair of vertices. Formally, a multigraph $G = (V, E)$ consists of a vertex set V and a multiset E of unordered pairs from V , where the same pair $\{u, v\}$ may appear multiple times. Similarly, a *multidigraph* (or *directed multigraph*) $D = (V, A)$ consists of a vertex set V and a multiset A of ordered pairs from $V \times V$, allowing multiple arcs from one vertex to another. The *multiplicity* of an edge $\{u, v\}$ or an arc (u, v) is the number of times it appears in E . Note that (u, v) and (v, u) are considered distinct arcs and each may have its own multiplicity.

A *weighted graph* is a graph $G = (V, E)$ together with a *weight function* $\omega : E \rightarrow \mathbb{R}$ (or more generally, $\omega : E \rightarrow S$ for some set S). For each edge $e \in E$, the value $\omega(e)$ is called the *weight* of edge e . Similarly, a *weighted directed graph* (or *weighted digraph*) is a directed graph $D = (V, A)$ together with a weight function $\omega : A \rightarrow \mathbb{R}$. These concepts extend naturally to multigraphs and multidigraphs, where the weight function assigns a weight to each instance of an edge or arc. Weighted graphs are fundamental in modeling real-world scenarios where edges represent connections with associated costs, distances, capacities, or other quantitative attributes.

A *vertex-colored graph* is a tuple $(G, \mathcal{C})$, where $\mathcal{C} = \{C_1, \dots, C_c\}$ is a set of *colors*, and each $C_i \subseteq V(G)$. An *edge-colored graph* is a tuple $(G, \mathcal{C})$, where $\mathcal{C} = \{C_1, \dots, C_c\}$ is a set of *colors*, and each $C_i \subseteq E(G)$. Note that each element may have zero or multiple colors. For each $v \in V(G)$ we write $\mathcal{C}(v)$ to denote the set of colors assigned to v . For each $e \in E(G)$ we write $\mathcal{C}(e)$ to denote the set of colors assigned to e . A colored graph is *properly* colored if every vertex has exactly one color and adjacent vertices (or edges in the case of an edge-colored graph) have different colors.

A set $S \subseteq V(G)$ ($S \subseteq E(G)$, respectively) in a vertex-colored graph (edge-colored graph, respectively) where every vertex (edge, respectively) has exactly one color is called a *rainbow* set if every element of the set has a distinct color.

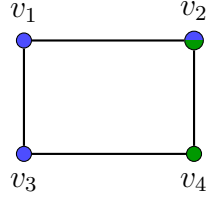


Figure 3.3: A vertex-colored graph where v_2 has multiple colors (blue and green), demonstrating that vertices can belong to multiple color classes. Here $\mathcal{C}(v_2) = \{C_1, C_2\}$ where C_1 is green and C_2 is blue.

Colors will serve multiple purposes in this thesis; sometimes they will be part of a problem input (such as when we use logic), other times they will provide a mechanism for algorithmic techniques (such as when we use color coding), or simply organize the input data in a convenient way.

Fundamental Structures

Several basic structures appear repeatedly in our analysis. In a graph, a *path* is a sequence of vertices $v_1, v_2, \dots, v_\ell$ where consecutive vertices are adjacent ($v_i v_{i+1} \in E(G)$ for all $i \in [\ell - 1]$) and all vertices (except possibly v_1 and v_ℓ) are distinct. In a digraph, a *directed path* is a sequence of vertices $v_1, v_2, \dots, v_\ell$ where consecutive vertices are connected by arcs following the direction: $(v_i, v_{i+1}) \in A(D)$ for all $i \in [\ell - 1]$, and all vertices (except possibly v_1 and v_ℓ) are distinct. The *length* of the path is $\ell - 1$ (the number of edges or arcs). The vertices v_1 and v_ℓ are the *endpoints* of the path, and the edges $v_1 v_2$ and $v_{\ell-1} v_\ell$ are the *endpoint edges* of the path. A (directed) *cycle* is a sequence of vertices $v_1, v_2, \dots, v_\ell$ where consecutive vertices are adjacent ($v_i v_{i+1} \in E(G)$ for all $i \in [\ell - 1]$), $v_1 = v_\ell$ and all intermediate vertices are distinct. See Figure 3.4.

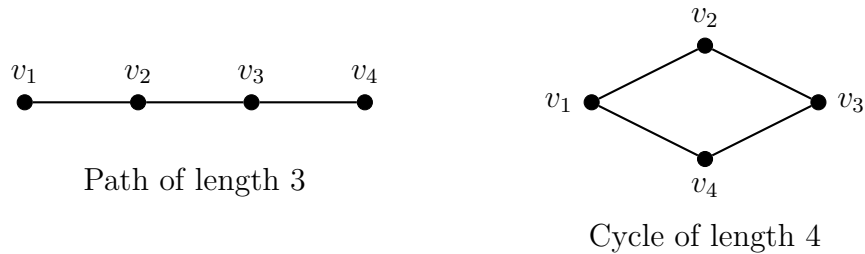


Figure 3.4: Examples of a path and a cycle.

The *distance* $d(u, v)$ between the vertices u and v is the length of a shortest (directed, in the case of a digraph) path between them, which we assume to be ∞ if no path exists. We define the *distance- r neighborhood* $N^r(v)$ of a vertex $v \in V(G)$ as the set that contains all vertices at distance at most r from v , excluding v itself. Furthermore, $N^r[v] = N^r(v) \cup \{v\}$. For sets, $N^r(S) = \bigcup_{v \in S} N^r(v) \setminus S$ and $N^r[S] = N^r(S) \cup S$.

A (di)graph is *connected* if a (directed) path exists between each pair of its vertices. A *connected component* of a (di)graph G is a maximal set of vertices $C \subseteq V(G)$ such that every pair of vertices in C is connected (in the underlying undirected graph in the case of a digraph) by a path. A *tree* is a connected graph that contains no cycles. A *directed tree* is a connected digraph whose underlying undirected graph is a tree. We call vertices of degree one in a tree the *leaves*. A *star* is a tree with one *central vertex* and neighboring leaves. A (directed) *forest* is a (di)graph in which each connected component is a (directed) tree. See Figure 3.5. We say that a graph is *acyclic* when it does not have cycles. When a graph has an edge between each pair of its vertices, then it is a *clique*. A clique on n vertices is denoted by K_n .

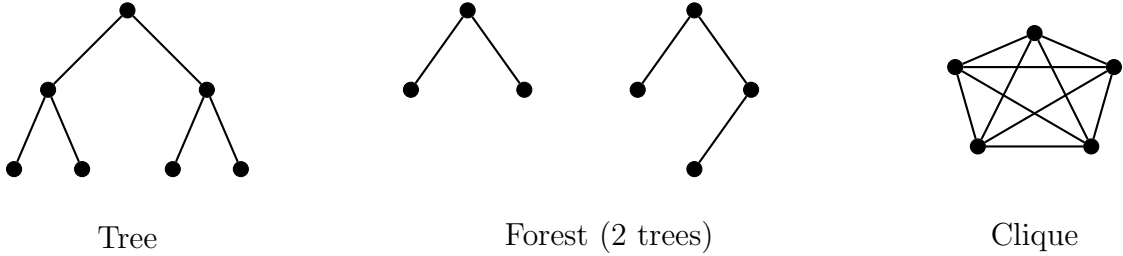


Figure 3.5: Examples of a tree, a forest, and a clique.

We manipulate graphs in multiple ways within the proofs. For example, we frequently use induced subgraphs. A *subgraph* G' of a graph G is a graph whose vertex set is a subset of $V(G)$ and whose edge set is a subset of $E(G)$ that only includes edges whose two endpoints are in $V(G')$. For $X \subseteq V(G)$, *induced subgraph* $G[X]$ has a vertex set X and an edge set $\{uv \in E(G) : u, v \in X\}$. For $X \subseteq V(D)$, *induced subgraph* $D[X]$ has a vertex set X and an arc set $\{uv \in A(D) : u, v \in X\}$. For $X \subseteq E(G)$, *induced subgraph* $G[X]$ has as edge set X and as vertex set the vertices of X . For $X \subseteq A(D)$, *induced subgraph* $D[X]$ has as arc set X and as vertex set the set of vertices of X . We also define $G - X = G[V(G) \setminus X]$ as the graph obtained by removing vertices from the vertex set X and all incident edges (arcs). See Figure 3.6. When $X \subseteq E(G)$ (or $X \subseteq A(D)$, respectively), $G - X$ ($D - X$, respectively) is the graph obtained by removing all edges (arcs, respectively) in X .

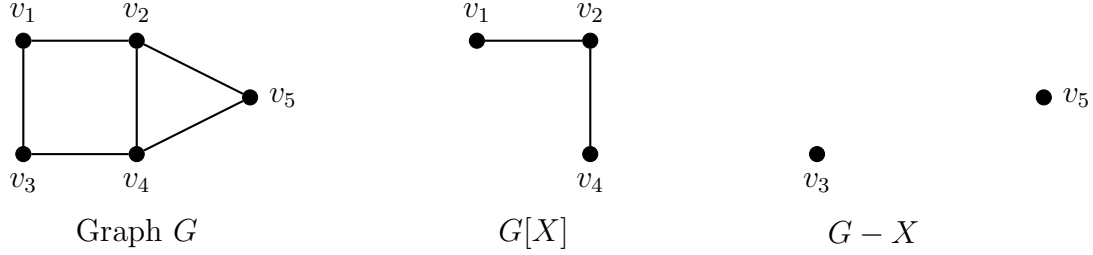


Figure 3.6: Induced subgraph and vertex deletion for $X = \{v_1, v_2, v_4\}$.

Some Graph Classes

A *graph class* is a set of graphs that share a common property or can be characterized by a specific condition. In this subsection, we explain the graph classes that appear later in the thesis. Algorithmic behavior can differ sharply across these classes. When the containment relation between the graph classes is recorded, tractability is downward-hereditary while intractability is upward-hereditary. A full complexities classification by graph class is beyond our scope; however, we do mention the types of graphs on which we prove hardness results.

Bipartite and Biclique-free Classes of Graphs. A graph G is *bipartite* if its vertex set can be partitioned into two disjoint sets A and B such that each edge in $E(G)$ has one endpoint in A and one in B . We call A and B the *partite sets* (or *parts*), and the pair (A, B) is called a *bipartition* of G . See Figure 3.7.

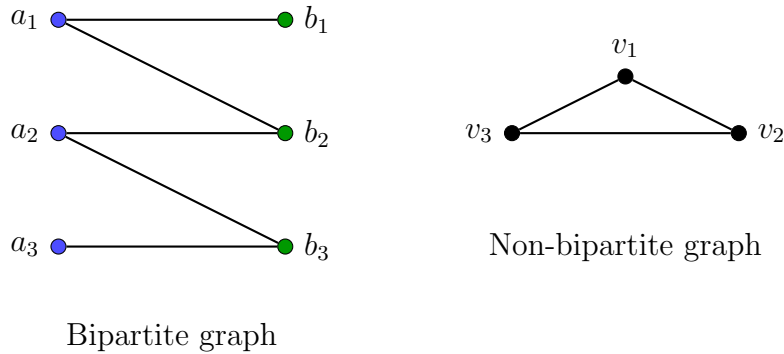


Figure 3.7: A bipartite graph with partite sets $A = \{a_1, a_2, a_3\}$ (blue) and $B = \{b_1, b_2, b_3\}$ (green) and a non-bipartite graph with an odd-length cycle.

Bipartite graphs appear prominently in our hardness results. In Chapter 5, we show that even on bipartite graphs, all of MATCHING-DISCOVERY, VERTEX/EDGE CUT-DISCOVERY, and SHORTEST PATH-DISCOVERY remain NP-hard.

If all edges between sets A and B are present in G we say that G is a *biclique*. When $|A| = a$ and $|B| = b$, we denote a biclique with partite sets A and B by $K_{a,b}$. A graph G is said to be *d-biclique-free* if it does not have the biclique $K_{d,d}$ as a subgraph.

Definition 1 (Biclique-free Class of Graphs). *A class C of graphs is biclique-free if there exists an integer d such that for all $G \in C$, G is d -biclique-free.*

Degenerate Graphs. A graph G is *d-degenerate* if every non-empty (that is, with at least one vertex) induced subgraph of G has a vertex of degree at most d . Equivalently, the vertices of G can be ordered $v_1, v_2, \dots, v_n$ so that each v_i has at most d adjacent vertices among $\{v_{i+1}, \dots, v_n\}$. The *degeneracy* of G , denoted $\text{dege}(G)$, is the smallest such d .

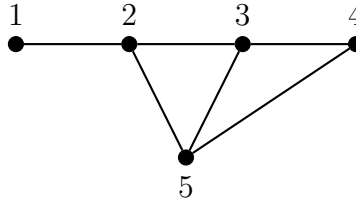


Figure 3.8: A 2-degenerate graph with degeneracy ordering shown using numbers. Each vertex has at most two adjacent vertices that appear “later” in the ordering.

The class of d -degenerate graphs is one proper subclass of the class of $d+1$ -biclique-free graphs and degeneracy also provides one natural measure of how sparse a graph is. Trees and forests are 1-degenerate, while the class of bipartite graphs has unbounded degeneracy. In Chapter 5, we often prove that our solution discovery problems (with polynomially solvable underlying properties, such as SHORTEST PATH-DISCOVERY) remain difficult even on graphs with very low degeneracy, demonstrating that the computational hardness is not due to dense graph structures.

Nowhere Dense Classes of Graphs: One Other Sparsity Measure

A graph H is a *minor* of a graph G , denoted $H \preceq G$, if there exists a mapping that associates each vertex v of H with a non-empty connected subgraph G_v of G such that $V(G_u)$ and $V(G_v)$ are disjoint for $u \neq v$ and whenever there is an edge between u and v in H ,

there is an edge between a vertex of G_u and a vertex of G_v . The subgraph G_v is referred to as the *branch set* of v . The *radius* of a connected graph (or subgraph) is the minimum, over all vertices, of the maximum distance from that vertex to any other vertex in the graph (or subgraph). We call H a *depth- r minor* of G , denoted $H \preceq_r G$, if each branch set has radius at most r .

Definition 2 (Nowhere Dense Class of Graphs). *A class $\mathcal{C}$ of graphs is nowhere dense if, for every $r \in \mathbb{N}$, there exists a function $t : \mathbb{N} \rightarrow \mathbb{N}$ such that $K_{t(r)} \not\preceq_r G$ and all $G \in \mathcal{C}$.*

From the definition, every nowhere dense class of graphs excludes at least one biclique.

3.2 Complexity Theory

Understanding the computational landscape of our problems requires us to have a solid foundation in complexity theory, both classical and parameterized. We start with classical complexity theory, which analyzes the computational resources needed to solve problems as a function of the input size.

Classical Complexity

We begin with formal definitions of the fundamental classical (time) complexity classes. Let Σ be a finite alphabet (typically $\{0, 1\}$). We denote by Σ^* the set of all finite strings over Σ , including the empty string. A *language* is a subset $L \subseteq \Sigma^*$. A (*classical*) *problem* is the task of determining whether a given string (equivalently, *instance*, *input*) $x \in \Sigma^*$ belongs to a language L . If $x \in L$ has size n , then a *solution* (or *witness*) for x is a proof of size polynomial in n that demonstrates that $x \in L$ (that is, x is a yes-instance of the relevant problem).

The encoding of problem instances is crucial for complexity analysis. For example, when encoding integers, we typically use binary representation unless otherwise specified. Some problems become artificially easier when integers are encoded in *unary* (where the integer n is represented by n consecutive 1s), since this inflates the input size. We mention such a problem in Chapter 6.

The complexity class $\mathcal{P}$ contains each problem that has an algorithm that given any input $x \in \Sigma^*$ of size n , returns a YES if $x \in L$ and NO if $x \notin L$ in time polynomial in n (at most $c' \cdot n^c$ time or equivalently, $\mathcal{O}(n^c)$ for some constants c', c). We say in this

case that the algorithm *decides* or “solves” the problem in polynomial time. Problems in P are considered efficiently solvable, or *tractable*. The class NP contains problems whose solutions can be checked in time polynomial in the size of the input. Specifically, if the answer for an instance x (of size n) of a problem is YES, then there exists a solution of size polynomial in n (at most $c' \cdot n^c$ size or equivalently, $\mathcal{O}(n^c)$ for some constants c', c) that can be checked in time polynomial in n to confirm the answer is indeed YES.

Hereafter, we say polynomial time to mean time polynomial in the size of the input, or polynomial size to mean size polynomial in the size of the input. We also say that an algorithm is a *polynomial time algorithm* if, on an input x of size n , it completes its task in time $\mathcal{O}(n^c)$ for some constant c .

Clearly, $P \subseteq NP$, but whether $P = NP$ remains one of the most important open problems in computer science. That is the case because, while checking a solution can be done in polynomial time, finding one can still be computationally hard.

A problem A is *polynomial-time reducible* to a problem B , denoted $A \leq_P B$, if instances of A can be transformed (that is, reduced) into instances of B in polynomial time so that the transformation preserves the answer (yes-instances map to yes-instances, no-instances map to no-instances). Intuitively, this means that B is at least as hard as A , since we can solve A by transforming it to B and solving B .

We call this polynomial-time transformation a polynomial-time *reduction*. In reductions, we employ *gadgets*, which are constructed substructures in an instance of problem B that simulate or encode the behavior of specific components from an instance of problem A .

A problem is *NP-hard* if every problem in NP is polynomial-time reducible to it. NP-hard problems are at least as hard as the hardest problems in NP . A problem is *NP-complete* if it is both NP-hard and in NP . NP-hard problems are *classically intractable*. The standard technique for proving that a problem B is NP-hard is to select another known NP-hard problem A and show that $A \leq_P B$.

Since most of our problems are classically computationally NP-hard as stated in the background and as we show later, it becomes imperative to explore methodologies beyond classical approaches for designing algorithms. One such avenue is the consideration of parameterized algorithms under the parameterized complexity theory; we present the preliminaries on this theory in what follows.

Parameterized Algorithms and Complexity

Through the framework developed by Downey and Fellows [69], we first define the general notion of a *parameterized problem* (that is, a parameterized variant of a classical problem) and then introduce some of the relevant parameterized time complexity classes, namely FPT, Para-NP and XP. A parameterized problem is the task of determining whether a given input (x, κ) belongs to a language $L \subseteq \Sigma^* \times \mathbb{N}$. For instance $(x, \kappa) \in \Sigma^* \times \mathbb{N}$, κ is called *parameter*.

Definition 3 (FPT and XP). *A parameterized problem is fixed-parameter tractable (or equivalently, in FPT) if it can be solved by an algorithm in time $f(\kappa) \cdot n^{O(1)}$ where $|x| = n$ and $f(\cdot)$ is a computable function. A parameterized problem is in XP if it can be solved in time $g(\kappa) \cdot n^{f(\kappa)}$ where $g(\cdot)$ is also a computable function. Clearly, $FPT \subseteq XP$, and this containment is believed to be strict.*

We say that an algorithm is a *fixed-parameter tractable algorithm* if, on an input (x, κ) where $|x| = n$, it completes its task in time $f(\kappa) \cdot n^{O(1)}$. One way to show that a parameterized problem is in FPT is to show that it has a *kernel* [48, 69].

Definition 4 (Kernelization and Kernel). *Kernelization or simply kernel is a fixed-parameter tractable algorithm that maps an instance (x, κ) to an instance (x', κ') , such that:*

- i. (x, κ) is a yes-instance if and only if (x', κ') is a yes-instance,
- ii. $\kappa' \leq \kappa$, and
- iii. $|x'| \leq f(\kappa)$ for some computable function $f(\cdot)$.

We have a polynomial kernel when f is a polynomial function.

Unfortunately, as we said before, not every parameterized problem that is in FPT admits a polynomial kernel. Under the complexity-theoretic assumption that $NP \not\subseteq coNP/P$ (where $coNP$ contains all problems where when the answer to an instance is NO, then there exists a certificate of polynomial size that can be checked in polynomial time), we can rule out the existence of a polynomial kernel for certain fixed-parameter tractable problems. The machinery for such lower bounds heavily relies on composing instances that are equivalent according to a *polynomial equivalence relation* [58].

Definition 5 (Polynomial Equivalence Relation). *An equivalence relation $\mathcal{R}$ on the set of instances of a problem A is called a polynomial equivalence relation if the following two conditions hold.*

1. There is an algorithm that, given two instances x and y of A decides whether x and y belong to the same equivalence class in time polynomial in $|x| + |y|$.
2. For any finite set S of instances of A , the equivalence relation $\mathcal{R}$ partitions the elements of S into at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ equivalence classes.

We can compose equivalent instances in more than one way. In this thesis, we focus on or-cross-compositions.

Definition 6 (Or-cross-composition [35]). *Let A be a problem, and let B be a parameterized problem. We say that A or-cross-composes into B if there is a polynomial equivalence relation $\mathcal{R}$ on the set of instances of A and an algorithm that, given t instances (where $t \in \mathbb{N}_{>0}$) $x_1, x_2, \dots, x_t$ belonging to the same equivalence class of $\mathcal{R}$, computes an instance (x, κ) in time polynomial in $\sum_{i=1}^t |x_i|$ such that the following properties hold.*

1. (x, κ) is a yes-instance of B if and only if there exists at least one index i such that x_i is a yes-instance of A .
2. κ is bounded above by a polynomial in $\max_{i=1}^t |x_i| + \log t$.

The inclusion $\text{NP} \subseteq \text{coNP}/\text{P}$ holds if an NP-hard problem A or-cross-composes into a parameterized problem B that admits polynomial kernels. As this inclusion is believed to be false, we will constantly make use of the following theorem to show that the existence of polynomial kernels is unlikely.

Theorem 7 ([35]). *If a problem A is NP-hard and A or-cross-composes into the parameterized problem B , then there are no polynomial kernels for B unless $\text{NP} \subseteq \text{coNP}/\text{P}$.*

On an unrelated note, an XP-time algorithm for an NP-hard problem does not imply $\text{P} = \text{NP}$, as the parameter κ is not fixed (that is, not a constant). Problems that remain NP-hard even when the parameter is a fixed integer (above some threshold) are the *Para-NP-hard* parameterized problems. For such problems, one cannot hope for algorithms running in FPT or even XP time.

To distinguish between parameterized problems solvable in XP time and parameterized problems solvable in fixed-parameter tractable time (that is, that have more efficient algorithms), the *W-hierarchy* of parameterized time complexity classes was introduced. This hierarchy is a subset of XP. A parameterized problem is *W[i]-hard* when every problem in $W[i]$ reduces to it via an *FPT-reduction* (the definition of each class $W[i]$ is omitted as it is not needed).

An FPT-reduction from a parameterized problem A to a parameterized problem B is an algorithm that transforms any input (x, κ) to A where $|x| = n$ to an input (x', κ') to B in time $f(\kappa) \cdot n^{O(1)}$ such that (x, κ) is a yes-instance of A if and only if (x', κ') is a yes-instance of B , and $\kappa' = g(\kappa)$ for some computable function g .

A problem that is $W[i]$ -hard for some $i \geq 1$ is *intractable*. On a positive note, FPT-reductions can also be used as an algorithmic tool.

A problem that is $W[i]$ -hard for all $i \geq 1$ is *XNLP-hard* [34]. One way to show that a parameterized problem B is XNLP-hard is to choose another parameterized problem A that is also XNLP-hard and perform a *pl-reduction* (equivalently, a *parameterized log-space reduction*) from A to B . A *pl-reduction* from A to B is an FPT-reduction that additionally computes (x', κ') using $\mathcal{O}(h(\kappa) + \log |x|)$ working space where h is a computable function.

3.3 Graph Problems

In our framework, we employ several graph problems; some serve as sources for reductions, others provide algorithms that we use as subroutines, and some define our property Π . We focus on the last category in this section of the preliminaries.

Shortest Path

A *shortest path* between vertices u and v in a graph G is a path from u to v with the minimum number of edges. The SHORTEST PATH problem is defined as follows.

SHORTEST PATH

Input: A graph G , vertices $s, t \in V(G)$, and an integer κ .

Output: YES if $d(s, t) = \kappa$. NO otherwise.

On a side note, as an example of subroutines, here we mention Breadth-First Search and Dijkstra's algorithms. *Breadth-First Search (BFS)* computes $d(s, v)$ for all vertices v in $O(n + m)$ time, and can then reconstruct a shortest path from s to each vertex v in worst-case $O(n)$ time. Throughout this thesis, whenever we need to calculate distances or paths between polynomially many pairs of vertices, we employ BFS, which runs in polynomial time. In *non-negatively weighted (di)graphs*, each edge (or arc for digraphs) e has a weight $w(e) \geq 0$, and the length of a path is the sum of its edge (arc for digraphs)

weights. While BFS no longer suffices for weighted graphs, Dijkstra's algorithm [65] finds shortest weighted paths from a single source vertex s in $O((n + m) \log n)$ time.

Matching

A *matching* in a graph G is a set $M \subseteq E(G)$ such that no two edges in M share one endpoint. A vertex v is *saturated* by a matching M if some edge in M is incident to v ; otherwise, v is *unsaturated*. See Figure 3.9a. The *size* of a matching is $|M|$, the number of edges it contains.



Figure 3.9: A matching $M = \{v_1v_2, v_3v_4\}$ shown in (a) has saturated vertices v_1, v_2, v_3, v_4 while v_5, v_6 are unsaturated. The path $v_6v_3v_4v_5$ is an M -augmenting path. Using this augmenting path, we obtain the maximum matching $M' = \{v_1v_2, v_6v_3, v_4v_5\}$ shown in (b), which saturates all vertices.

A matching M is *maximum* if no other matching in G contains more edges than M . An *M -augmenting path* is a path that starts and ends at unsaturated vertices and alternates between edges not in M and edges in M . Notice how one can obtain a larger matching by removing from M each edge in M that appears in the M -augmenting path and adding to M each edge in the M -augmenting path that is not in M . See Figure 3.9b. The following fundamental theorem of Berge characterizes a maximum matching.

Theorem 8 ([18]). *A matching M in a graph G is maximum if and only if G has no M -augmenting path.*

This characterization leads to polynomial-time algorithms for finding maximum matchings in graphs (since finding an M -augmenting path in a graph G can be done in $\mathcal{O}(m)$ time with the help of BFS) and will play a crucial role in our complexity analysis of MATCHING-DISCOVERY. The classical MATCHING problem is defined as follows.

MATCHING

Input: A graph G , and an integer κ .

Output: YES if there exists a matching $M \subseteq E(G)$ such that $|M| \geq \kappa$. NO otherwise.

Vertex and Edge Cuts

Given vertices s and t in a graph G , a *vertex cut* (or an *s - t vertex cut*) is a set $C \subseteq V(G) \setminus \{s, t\}$ such that every path from s to t in G contains a vertex in C . Equivalently, removing C disconnects s from t in the resulting graph. An *edge cut* (or an *s - t edge cut*) is a set $C \subseteq E(G)$ such that every path from s to t contains an edge in C . See Figure 3.10. A *line graph* of a graph G is a graph $L(G)$ where each vertex of $L(G)$ represents an edge of G , and two vertices in $L(G)$ are adjacent if and only if their corresponding edges in G share a common endpoint. See Figure 3.10c.

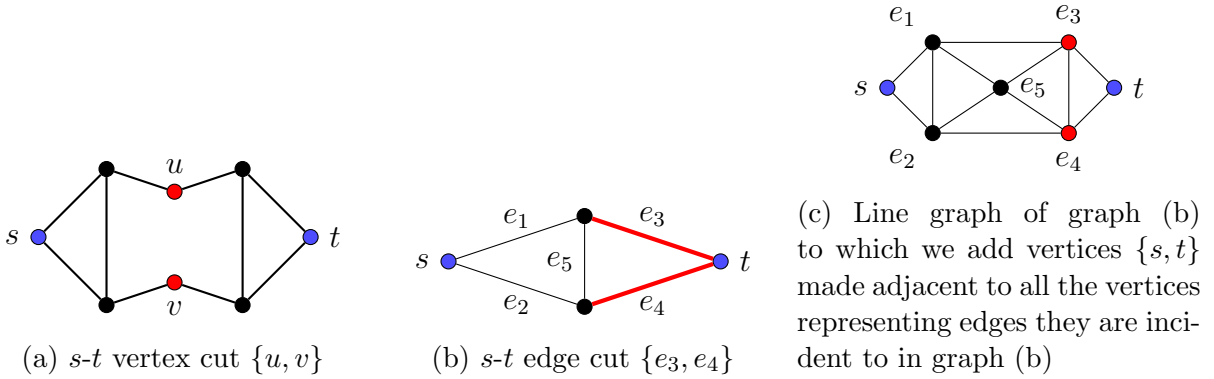


Figure 3.10: Examples of vertex and edge cuts separating s from t . The s - t edge cut $\{e_3, e_4\}$ in (b) corresponds to the vertex cut $\{e_3, e_4\}$ in the line graph drawn in (c).

It is easy to see that if C is an s - t edge cut in a graph G , then the vertices representing the edges in C form an s - t vertex cut in the augmented line graph constructed from $L(G)$ by adding two new vertices s and t , making s adjacent to all vertices in $L(G)$ that represent edges incident to s in G , and making t adjacent to all vertices in $L(G)$ that represent edges incident to t in G [179]. The size of a cut is the number of vertices (for vertex cuts) or edges (for edge cuts) it contains. The classic VERTEX/EDGE CUT problem is defined as follows.

VERTEX/EDGE CUT

Input: A graph G , two vertices $s, t \in V(G)$, and an integer κ .

Output: YES if there exists an s - t vertex/edge cut of size at most κ . NO otherwise.

Independent Set and Vertex Cover

An *independent set* in a graph G is a set $I \subseteq V(G)$ such that no two vertices in I are adjacent. See Figure 3.11. The INDEPENDENT SET problem is defined as follows.

INDEPENDENT SET

Input: A graph G and an integer κ .

Output: YES if there exists an independent set $I \subseteq V(G)$ such that $|I| \geq \kappa$. NO otherwise.

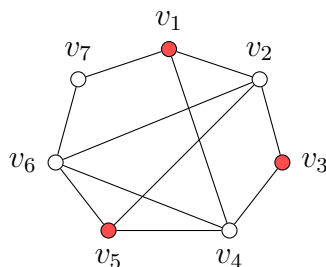


Figure 3.11: An independent set $I = \{v_1, v_3, v_5\}$ (shown in red). Note that no two vertices in I are adjacent, while the remaining vertices form a vertex cover.

A *vertex cover* in a graph G is a set $C \subseteq V(G)$ such that every edge in $E(G)$ has at least one endpoint in C . Equivalently, C is a vertex cover if and only if $V(G) \setminus C$ is an independent set. We say that the vertex cover C *covers* G . When $u \in C$, then uc is *covered* and u *covers* uc . The *vertex cover number* is the size of a minimum vertex cover in G . The following is the definition of the VERTEX COVER problem.

VERTEX COVER

Input: A graph G and an integer κ .

Output: YES if there exists a vertex cover $C \subseteq V(G)$ such that $|C| \leq \kappa$. NO otherwise.

Dominating Set

A *dominating set* in a graph G is a set $D \subseteq V(G)$ such that every vertex in $V(G) \setminus D$ is adjacent to at least one vertex in D . We say that the dominating set D *dominates* G . When $u \in D$, we say that every vertex $v \in V(G)$ adjacent to u is *dominated* and that u (or D) *dominates* v . Here is the definition of the DOMINATING SET problem.

DOMINATING SET

Input: A graph G and an integer κ .

Output: YES if there exists a dominating set $D \subseteq V(G)$ such that $|D| \leq \kappa$. NO otherwise.

An important concept that was introduced by Dawar and Kreutzer to approach domination problems [60] is *domination cores*.

Definition 9. Let G be a graph and $k \geq 1$. A set $C \subseteq V(G)$ is a k -domination core if every set of size at most k that dominates C also dominates G .

3.4 Solution Discovery Problems

In this thesis, a *configuration* in a graph G is a subset of its vertices (or edges). In all of our solution discovery problems given a graph G and an *initial configuration* $S \subseteq V(G)$ (or $S \subseteq E(G)$) where $|S| = k$, we place one token on each element of S , and our goal is to move these tokens into a *final configuration* $T \subseteq V(G)$ (or $T \subseteq E(G)$) where each element contains at most one token and that satisfies a certain property Π , using at most a *budget* of b token slides. In a single *token slide*, a token can move (or specifically, *slide*) from a vertex to an adjacent vertex (an edge to an adjacent edge). The sequence of token slides that we perform to get from an initial configuration S to a final configuration T is called a *transformation sequence* $\vec{T}$. We say that in a solution $(T, \vec{T})$ to a solution discovery problem, a token t initially at a vertex $u \in S$ is *destined* for a vertex $v \in T$ when $\vec{T}$ includes a subsequence of token slides that slide the token from u to v and no other token slide in $\vec{T}$ slides t . Note that only S and T are sets; during the transformation, multiple tokens can be placed at the same vertex (or edge) and this is equivalent to the model where no vertex can have more than one token during the transformation.

When Π is a fixed property defined by a classical problem (such as SHORTEST PATH, that is, the property is “the k tokens form a shortest path between s and t ”), we get the

Π -DISCOVERY problem (such as the SHORTEST PATH-DISCOVERY problem). When Π is a fixed property defined by a logical formula, we obtain a particularly expressive framework. Let $\phi(X)$ be a formula with a free vertex-set variable X that expresses a property of colored graphs. In the ϕ -DISCOVERY problem, the final configuration must satisfy the formula; we seek a set T of vertices ($|T| = |S|$) such that $G \models \phi(T)$. Formulas can capture many classical problems: for instance, “ X forms an s - t vertex cut” can be expressed as a logical formula. In the most general version, called MSO-DISCOVERY (MONADIC SECOND-ORDER-DISCOVERY) or FO-DISCOVERY (FIRST-ORDER-DISCOVERY), the formula (MSO- or FO-formula, respectively) $\phi(X)$ itself is part of the input. Since MSO is more expressive than FO, the tractability of MSO-DISCOVERY when parameterized by some parameter implies the tractability of FO-DISCOVERY when parameterized by the same parameter, while the intractability of FO-DISCOVERY when parameterized by some parameter implies intractability of MSO-DISCOVERY when parameterized by the same parameter.

For the purpose of analysis of these problems, we define $b(u, v)$ as the minimum number of token slides needed to move a token initially placed on some vertex (edge) $u \in S$ to (an incident edge to) the vertex v . Note that in a yes-instance, we can always assume $b \leq kn$ where b is the budget and k is the number of tokens in S , since each token needs to traverse a path of length at most n to reach its final position. Hence, we have the following.

Claim 10. *Let b be the budget for a solution discovery problem instance with n vertices and $|S| = k$, then we can assume that $b \leq kn$.*

This bound ensures that all solution discovery problems that we consider are in NP since we can give a transformation sequence of polynomial size as a solution.

Finally, we say that a vertex $u \in V(G)$ (an edge $e \in E(G)$, respectively) is *irrelevant with respect to* $v \in S$ (*irrelevant with respect to* $e' \in S$, respectively) if a given a solution $(T, \vec{T})$ and where the token on v (e' , respectively) slides and stays on the vertex u (edge e , respectively), we can construct a final configuration T' that requires the same number of token slides as in $\vec{T}$ and where $u \notin T'$ ($e \notin T'$, respectively).

Problem Definitions

The following are the solution discovery problems that we study in this thesis.

INDEPENDENT SET DISCOVERY

Input: A graph G , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .
Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into an independent set $I \subseteq V(G)$ ($|I| = k$) such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

DOMINATING SET DISCOVERY

Input: A graph G , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .
Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a dominating set $D \subseteq V(G)$ ($|D| = k$) such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

VERTEX COVER DISCOVERY

Input: A graph G , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .
Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a vertex cover $C \subseteq V(G)$ ($|C| = k$) such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

MATCHING DISCOVERY

Input: A graph G , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .
Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a matching $M \subseteq E(G)$ ($|M| = k$) such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

VERTEX/EDGE CUT DISCOVERY

Input: A graph G , vertices $s, t \in V(G)$, an initial configuration $S \subseteq V(G)/E(G)$ with $|S| = k$, and a budget b .
Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a vertex/edge cut $C \subseteq V(G)/E(G)$ ($|C| = k$) separating s and t such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

SHORTEST PATH DISCOVERY

Input: A graph G , vertices $s, t \in V(G)$, an initial configuration $S \subseteq V(G)$ with $|S| = k = d(s, t)$, and a budget b .

Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into the vertex set of a shortest s - t path in G such that the number of token slides in $\vec{T}$ is at most b . NO otherwise.

MSO-DISCOVERY

Input: A (colored) graph $(G, \mathcal{C})$, an MSO formula $\phi(X)$ with a free set variable X , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .

Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a set $T \subseteq V(G)$ ($|T| = k$) such that $G \models \phi(T)$ and the number of token slides in $\vec{T}$ is at most b . NO otherwise.

FO-DISCOVERY

Input: A (colored) graph $(G, \mathcal{C})$, an FO formula $\phi(X)$ with a free set variable X , an initial configuration $S \subseteq V(G)$ with $|S| = k$, and a budget b .

Output: YES if there exists a transformation sequence $\vec{T}$ that transforms S into a set $T \subseteq V(G)$ ($|T| = k$) such that $G \models \phi(T)$ and the number of token slides in $\vec{T}$ is at most b . NO otherwise.

3.5 Reconfiguration Problems

While solution discovery problems ask whether we can transform an initial configuration into *some* final configuration satisfying certain properties, our reconfiguration problems study the transformation between two *specified* configurations.

We say that k tokens are *labeled* when each token has a unique label from $[k] = \{1, 2, \dots, k\}$. A *labeled configuration* is a sequence $(v_1, v_2, \dots, v_k)$ where $v_i \in V(G)$ indicates that the token with the label i is placed on the vertex v_i . In our reconfiguration problems, we are given an initial (*labeled*) configuration S and a *target (labeled) configuration* T , and we must determine whether S can be transformed exactly into T within a given budget of b multi-slides. A *multi-slide* of a token from vertex $u \in V(G)$ to vertex $v \neq u \in V(G)$ moves the token along a path from u to v in a single move, provided that v and all intermediate

vertices on the path are token-free. For directed graphs, a multi-slide must follow a directed path. We may drop the word “labeled” and only say “configuration” when it is clear from context which configuration we are referring to.

In the context of our reconfiguration problems, a *transformation sequence* $\vec{M} = (m_1, m_2, \dots, m_\ell)$ is a sequence of multi-slides where each multi-slide $m_i = (s_i, t_i)$ is *valid*; that means, after executing the multi-slides $m_1, \dots, m_{i-1}$, there is a token on the *source vertex* s_i , no token on the *target vertex* t_i , and a token-free path from s_i to t_i . We call this path from s_i to t_i the *path for the multi-slide*. For a transforming sequence of multi-slides, we use $G_{\vec{M}}$ or $(D_{\vec{M}},$ for an instance with a directed graph) to denote the induced subgraph $G[X]$ ($D[X]$, respectively) where X is an edge set that contains all the edges of the (directed) paths for the multi-slides in $\vec{M}$. We define the *obstacles* to be $O = S \cap T$ (vertices that contain tokens in both configurations). We call a multi-slide whose s_i is in O a *clearing multi-slide*, and a one whose t_i is in O a *filling multi-slide*. A token on vertex $v \in V(G)$ *blocks* a multi-slide $m_i = (s_i, t_i)$ for some i if v lies on every path from s_i to t_i (excluding s and t themselves). In this case, m_i is *blocked by* v .

Problem Definitions

We define four reconfiguration problems based on whether tokens are labeled and whether the graph is directed.

UNLABELED UNDIRECTED TOKEN MOVING (UUTM)

Input: An undirected graph G , configurations $S \subseteq V(G)$ and $T \subseteq V(G)$ with $|S| = |T| = k$, and a budget b .

Output: YES if there exists a transformation sequence of size at most b that transforms S into T . NO otherwise.

UNLABELED DIRECTED TOKEN MOVING (UDTM)

Input: A directed graph D , configurations $S \subseteq V(D)$ and $T \subseteq V(D)$ with $|S| = |T| = k$, and a budget b .

Output: YES if there exists a transformation sequence of size at most b that transforms S into T . NO otherwise.

LABELED UNDIRECTED TOKEN MOVING (LUTM)

Input: An undirected graph G , labeled configurations $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_k)$ where $s_i, t_i \in V(G)$, and a budget b .

Output: YES if there exists a transformation sequence of size at most b that transforms S into T (where token i moves from s_i to t_i). NO otherwise.

LABELED DIRECTED TOKEN MOVING (LDTM)

Input: A directed graph D , labeled configurations $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_k)$ where $s_i, t_i \in V(D)$, and a budget b .

Output: YES if there exists a transformation sequence of size at most b that transforms S into T (where token i moves from s_i to t_i). NO otherwise.

3.6 Structural Graph Parameters

Under parameterized complexity, one can choose any “measurable” characteristic of the input to play the role of the parameter κ . For example, for a problem that encodes a graph G in its input, we can let κ be the maximum degree. Another characteristic can be one or more of various *structural graph parameters*. Of these, we present the ones relevant for our results.

Treewidth, Pathwidth and Their Decompositions

A *tree decomposition* of a graph G is a pair $(T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{B_i : i \in V(T)\}$ is a collection of subsets of $V(G)$ (called *bags*) such that (see Figure 3.12):

1. $\bigcup_{i \in V(T)} B_i = V(G)$ (every vertex appears in some bag),
2. for every edge $uv \in E(G)$, there exists $i \in V(T)$ with $\{u, v\} \subseteq B_i$ (every edge appears in some bag), and
3. for every vertex $v \in V(G)$, the set $\{i \in V(T) : v \in B_i\}$ induces a connected subgraph of T .

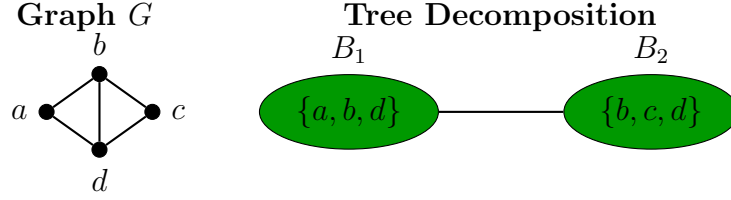


Figure 3.12: A graph G and its tree decomposition of width $\max(|B_i|) - 1 = 2$.

The *width* of a tree decomposition is $\max_{i \in V(T)} |B_i| - 1$. The treewidth of G , denoted $tw(G)$, is the minimum width over all tree decompositions of G . Treewidth measures how “tree-like” a graph is. Trees have treewidth one, while clique graphs of size n have treewidth $n - 1$. Computing a graph’s treewidth and its associated tree decomposition can be done in fixed-parameter tractable time when the parameter is $tw(G)$ [28].

A *path decomposition* of a graph G is a tree decomposition $(T, \mathcal{B})$ where T is a path. The pathwidth of G , denoted $pw(G)$, is the smallest width of any path decomposition of G . A *nice path decomposition* of G is one that begins and ends with nodes corresponding to empty bags and such that each other node in the decomposition corresponds to a bag that either *introduces* a vertex $v \in V(G)$ ($B_i = B_{i-1} \cup \{v\}$ for $v \notin B_{i-1}$) or *forgets* one ($B_i = B_{i-1} \setminus \{v\}$ for $v \in B_{i-1}$). Since any path decomposition is also a tree decomposition (given that a path is also a tree), we have that $tw(G) \leq pw(G)$ for any graph G .

Additional Structural Graph Parameters

Two vertices $u, v \in V(G)$ are *twins* if $N(u) = N(v)$ or $N[u] = N[v]$. The *neighborhood diversity* of a graph G , denoted by $nd(G)$, is the number of subsets in the unique partition of $V(G)$ into maximal sets of twin vertices called *twin classes*. See Figure 3.13a. It is known that the neighborhood diversity and the corresponding partition of a graph can be computed in linear time [144, 172]. Neighborhood diversity as a parameter provides a powerful tool for parameterized algorithmic design (as was demonstrated by Giménez et al. [99]), since vertices within the same twin class behave identically with respect to the rest of the graph.

A *twin cover* of a graph G is a set $M \subseteq V(G)$ whose deletion leaves a disjoint union of cliques such that the vertices of each clique are twins in G . The *twin cover number* of G , denoted by $tc(G)$, is the minimum size of a twin cover of G and can be computed in fixed-parameter tractable time when $tc(G)$ is the parameter [91]. See Figure 3.13b.

A *modulator to paths* is a set of vertices $X \subseteq V(G)$ such that $G - X$ is a disjoint

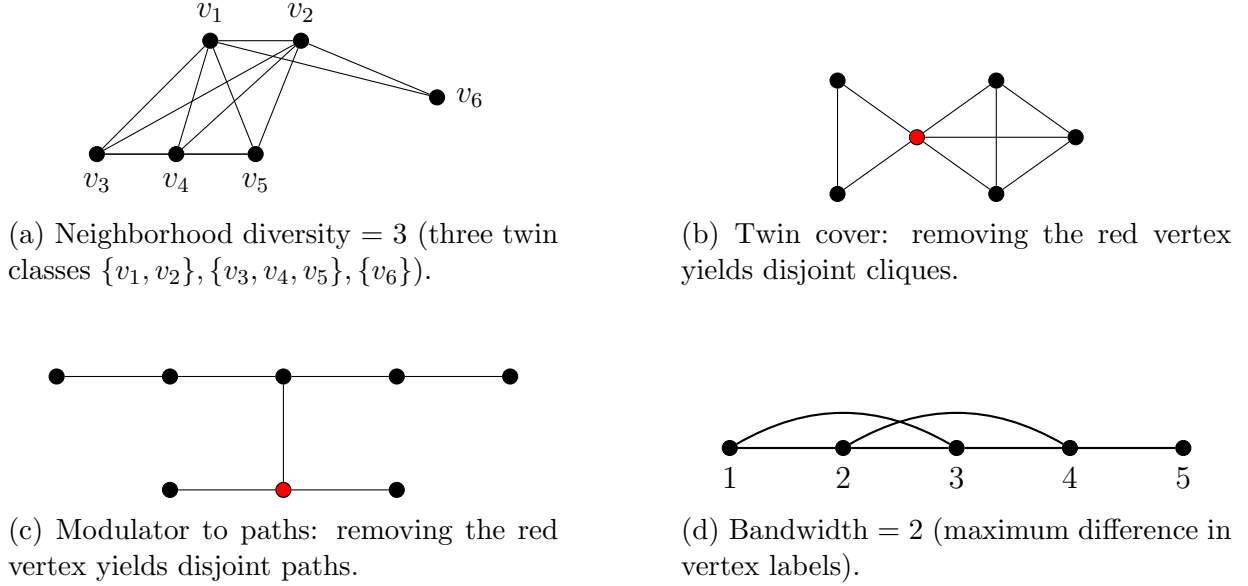


Figure 3.13: Additional structural graph parameters.

union of paths. The size of a minimum such modulator, denoted by $mp(G)$, is called *modulator to paths number*. See Figure 3.13c. Similarly, a *modulator to stars* is a set of vertices $X \subseteq V(G)$ such that $G - X$ is a disjoint union of stars. The size of a minimum such modulator, denoted by $ms(G)$, is referred to as *modulator to stars number*. It has been shown that one can compute a size- k modulator to paths or stars (if it exists) in fixed-parameter tractable time when k is the parameter [151].

The *bandwidth* of a graph G is the minimum integer $bw(G)$ such that there exists a bijection $f : V(G) \rightarrow \{1, \dots, |V(G)|\}$ satisfying that for every edge $\{u, v\} \in E(G)$, $|f(u) - f(v)| \leq bw(G)$. See Figure 3.13d. Computing bandwidth of a graph is NP-hard and W[1]-hard when parameterized by $bw(G)$ [32].

3.7 Algorithmic Techniques

Our algorithms rely on several key techniques to achieve efficiency. While color coding and derandomization have already been discussed, we present one additional technique here.

To obtain polynomial kernels for MATCHING-DISCOVERY and INDEPENDENT SET-DISCOVERY when parameterized by the number of tokens k , we employ the celebrated

Sunflower Lemma of Erdős and Rado [79], which has become a fundamental tool in parameterized complexity. The key insight is that, when dealing with large families of sets, redundancy will emerge in the form of a sunflower structure that will allow us to safely remove some of its elements.

Definition 11 (Sunflower). *A sunflower with p petals is a collection of sets $S_1, S_2, \dots, S_p$ such that the intersection of any pair of sets is the same, i.e., $S_i \cap S_j = C$ for all distinct $i, j \in [p]$. The set C is called the core of the sunflower, and the sets $S_i \setminus C$ are called the petals.*

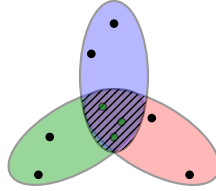


Figure 3.14: An example of a sunflower (with pink, blue and green petals) and a common core (the shaded intersection area). The black vertices represent elements that belong to individual petals but not to the core, while the green vertices represent the elements in the core that are shared by all three sets.

Lemma 12 (Sunflower Lemma [1]). *Let $\mathcal{A}$ be a family of sets (without duplicates) over a universe U , such that each set in $\mathcal{A}$ has cardinality at most d . If $|\mathcal{A}| > d!(p-1)^d$, then $\mathcal{A}$ contains a sunflower with p petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and p .*

In the context of transformation problems, the Sunflower Lemma provides a powerful algorithmic tool. For example, when we encounter a large family of sets (e.g., a large family of edges at a fixed token sliding distance from a token), the lemma guarantees the existence of a sunflower, which we can exploit in some way.

Chapter 4

Multi-Sliding

In this chapter, we focus on the results of the reconfiguration problems. In particular, we study the parameterized complexity of UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING with respect to the parameters k (the number of tokens) and b (the size of the transformation sequence). In addition, we focus on the parameterized complexity of LABELED UNDIRECTED TOKEN MOVING and LABELED DIRECTED TOKEN MOVING with respect to parameter b or $b + f$ where f represents the number of token-free vertices. The results presented in this chapter appear in the paper titled “*Parameterized Complexity of Reconfiguration of Atoms*”, in proceedings of *WALCOM: Algorithms and Computation 2022*, and co-authored with Alexander Cooper, Amer E. Mouawad, and Naomi Nishimura [52].

We start with the positive results achievable when the tokens are unlabeled. Thus, we first establish that UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING when parameterized by k are fixed-parameter tractable by showing that tokens move at most once in any minimum size transformation sequence. This enables us to construct kernels of size at most $2k$, and thus show fixed-parameter tractability. We then prove that UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING are also fixed-parameter tractable when parameterized by the budget b . For the undirected case, we characterize one optimal solution as a forest of minimum special trees which can be found in an input graph with existing efficient fixed-parameter tractable techniques. We follow a similar approach for the directed case; we first prove that the transformation sequence of one optimal solution induces a directed forest in the input graph and then employ color-coding techniques to find this forest structure in yes-instances.

In contrast to the unlabeled problems, we demonstrate that LABELED UNDIRECTED TOKEN MOVING and LABELED DIRECTED TOKEN MOVING are $W[1]$ -hard when parameterized by b or $b + f$ through reductions from the MULTICOLORED SUBGRAPH ISOMORPHISM which we define in Section 4.3.

4.1 Fixed-Parameter Tractability of Unlabeled Directed/Undirected Token Moving for Parameter k

In this section, we prove the following theorem.

Theorem 13. *UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING are fixed-parameter tractable when parameterized by k .*

We start by noting some observations that follow directly from the problem definitions. In the observations, we fix an instance of UNLABELED UNDIRECTED TOKEN MOVING or UNLABELED DIRECTED TOKEN MOVING and a corresponding transformation sequence $\vec{M}$ of multi-slides.

Observation 14. *Let $\ell = |\vec{M}|$. If a vertex $v \in V(G)$ ($v \in V(D)$, respectively) blocks a multi-slide $m_i = (s_i, t_i)$ in G (D , respectively) for some $i \in [\ell]$, then there exists a path (directed path, respectively) from v to t_i .*

Observation 15. *Let $\ell = |\vec{M}|$. Suppose that $m_\ell = (s_\ell, t_\ell)$ is the last multi-slide in a transformation sequence $\vec{M}$ from S to T . The sequence $\vec{M}'$ formed from $\vec{M}$ by removing m_ℓ is a transformation sequence from S to a configuration T' , where T' differs from T by having a token on s_ℓ instead of on t_ℓ .*

Claim 16. *Let $\ell = |\vec{M}|$. Suppose that $m_\ell = (s_\ell, t_\ell)$ is the last multi-slide in a transformation sequence $\vec{M}$ from S to T and let $\vec{M}'$ be formed from $\vec{M}$ by removing m_ℓ and T' be the configuration formed by applying $\vec{M}'$ starting from S in G (D , respectively). Suppose that there exists a transformation sequence $\vec{M}''$ from S to a configuration T'' , where T'' differs from T' by a single token, where T'' has a token on a vertex u and T' has a token on a vertex $v \neq s_\ell$. Then if m_ℓ is blocked after the execution of $\vec{M}''$ on S in G (D , respectively), there must be a token-free (directed, if in D) path from u to t_ℓ .*

Proof of Claim 16. We know that m_ℓ is valid in $\vec{M}$ and hence m_ℓ is valid after the execution of $\vec{M}'$ starting from S in G (D , respectively). The only difference between $\vec{M}''$ and $\vec{M}'$ is the presence of a token on u instead of v , so the only vertex that can block m_ℓ is u . The fact that u is the only reason that there are no free paths from s to t , combined with Observation 14, implies that there is a token-free (directed, respectively) path from u to t_ℓ . $\square$

Lemma 17. *For any instance (G, S, T, b) of UNLABELED UNDIRECTED TOKEN MOVING or instance (D, S, T, b) of UNLABELED DIRECTED TOKEN MOVING, there exists a shortest transformation sequence in which there is no token that multi-slides more than once.*

Proof of Lemma 17. We provide the proof for the directed case, which implies the undirected case. We assume to the contrary that there exists an instance for which any shortest transformation sequence requires the multi-slide of at least one token a second time. We minimize the length of the subsequence of multi-slides in $\vec{M}$ starting at the first multi-slide of a token and ending at the second multi-slide of the same token. We can represent $\vec{M}$ as $m_1, m_2, \dots, m_{\ell-1}, m_\ell$, where the multi-slide $m_i = (s_i, t_i)$. Since the first and last move are of the same token, $t_1 = s_\ell$.

To prove the lemma, it suffices to show that we can form a shorter transformation sequence of multi-slides, $\vec{M}'$, such that $\vec{M}$ and $\vec{M}'$ result in the same target configuration T and $\vec{M}'$ is shorter than $\vec{M}$. We will refer to multi-slides of the form (s_i, t_i) as *matching multi-slides* and multi-slides of the form (s_i, t_{i+1}) as *offset multi-slides*; a sequence of matching (offset) multi-slides consists of matching (offset) multi-slides for consecutive values of i .

We construct $\vec{M}'$ as a sequence of *segments*, where each segment is formed by adding zero or more valid offset multi-slides until a blocked offset multi-slide (s_j, t_{j+1}) is reached or $t_{j+1} = t_\ell$, adding all valid matching multi-slides starting at (s_{j+1}, t_{j+1}) until a blocked matching multi-slide $(s_{j'}, t_{j'})$ is reached, and then adding the *linking multi-slide* $(s_j, t_{j'})$. See Figure 4.1. For the next segment, we start by attempting to add the offset move $(s_{j'}, t_{j'+1})$. Note that $\vec{M}'$ cannot end with a matching multi-slide as that implies we can remove multi-slide (s_j, t_j) of the last segment (since the token on s_j does not block any multi-slide in $\vec{M}$) and obtain a smaller subsequence than $\vec{M}$.

To prove that the multi-slides in the segments are all valid, we first show that (s_1, t_2) , the first multi-slide in the first segment, is valid. We know that in $\vec{M}$, m_2 is valid after m_1 is executed, but since $\vec{M}$ is of minimum length, m_2 cannot be executed before m_1 . Since (s_1, t_1) results in a token-free path between s_2 and t_2 , we know that s_1 must contain the only token on an intermediate vertex on the path for the multi-slide m_2 from s_2 to t_2 , and hence there are token-free directed paths from s_1 to each of s_2 and t_2 .

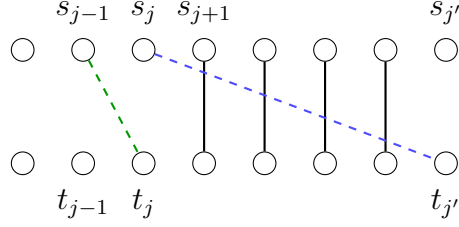


Figure 4.1: Formation of a segment in $\vec{M}'$; sources and target vertices for consecutive values of i are depicted by rows of circles, offset multi-slide by dashed green lines, matching multi-slides by solid black lines, and the linking multi-slide by a dashed blue line.

To show that all the multi-slides in $\vec{M}'$ are valid, we prove the following statements by induction on the number of segments of $\vec{M}'$ executed so far.

1. The first matching multi-slide in any segment is valid.
2. The linking multi-slide in any segment is valid.

To show that (1) holds, we suppose instead that the matching multi-slide (s_{j+1}, t_{j+1}) is blocked after the execution of (s_{j-1}, t_j) . By the definition of the construction of $\vec{M}'$, we are attempting to add multi-slide (s_{j+1}, t_{j+1}) because (s_j, t_{j+1}) is not valid. Because the matching move (s_{j+1}, t_{j+1}) is valid in $\vec{M}$, by Claim 16 (where (s_{j+1}, t_{j+1}) is m_ℓ and s_j is u), we can then conclude that s_j blocks the move, and that there is a token-free directed path from s_j to t_{j+1} , contradicting the assumption that (s_j, t_{j+1}) is not valid.

For part (2), we consider the placement of tokens after applying $\vec{M}$ and $\vec{M}'$ up to before $(s_{j'}, t_{j'})$ and $(s_j, t_{j'})$. By Claim 16 (where $(s_{j'}, t_{j'})$ is m_ℓ and s_j is u), the fact that the multi-slide $(s_{j'}, t_{j'})$ is valid in $\vec{M}$ but not in $\vec{M}'$ implies that there is a token-free directed path from s_j to $t_{j'}$, and hence the multi-slide $(s_j, t_{j'})$ is a valid multi-slide.

Finally, we complete the proof by showing that applying $\vec{M}$ and $\vec{M}'$ results in the same target configuration. By the end of the first $\ell - 1$ multi-slides in $\vec{M}'$, the only difference with the configuration obtained by the end of applying the first $\ell - 1$ multi-slides in $\vec{M}$ starting from S , is that there is no token on $t_1 = s_\ell$ and a token on t_ℓ . Thus, no further multi-slides are required in $\vec{M}'$ to achieve the configuration reached in $\vec{M}$ by the final multi-slide, (s_ℓ, t_ℓ) . $\square$

We use Lemma 17 to show that we can find in polynomial time a polynomial kernel for each instance of UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED

TOKEN MOVING whose vertices are from $S \cup T$ (at most $2k$). We use those polynomial kernels to form fixed-parameter tractable algorithms with respect to k .

Lemma 18. *Given an instance (G, S, T, b) of UNLABELED UNDIRECTED TOKEN MOVING (an instance (D, S, T, b) of UNLABELED DIRECTED TOKEN MOVING, respectively), we can, in polynomial time, find a kernel (G', S, T, b) ((D', S, T, b) , respectively) of size at most $2k$.*

Proof of Lemma 18. From Lemma 17, we know that there exists a shortest transformation sequence in which no token moves more than once. Thus, a vertex $v \notin S \cup T$ is never the source vertex nor the target vertex of a multi-slide in such a sequence, the only role it can play is to connect its neighbors. We form G' from G by adding an edge between each pair of neighbors of each such vertex v and deleting v . To form D' from D , we instead add an arc from each in-neighbor of v to each out-neighbor of v , where $v \notin S \cup T$ and delete v . $\square$

Corollary 19. *UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING are fixed-parameter tractable when parameterized by k .*

Proof of Lemma 19. To form the algorithm, we observe by Lemma 17 that it suffices to determine the set of obstacles O that are sources of multi-slides, the pairing of source vertices and target vertices into multi-slides, and the order of multi-slides. By Lemma 17 also, the number of multi-slides b will be at most $|S \setminus T| + |O|$, allowing us to try all possible choices of vertices in O , pairings, and orders in fixed-parameter tractable time with respect to parameter k (namely $\mathcal{O}(2^{k \log k}) \cdot n^{\mathcal{O}(1)}$). $\square$

4.2 Fixed-Parameter Tractability of Unlabeled Directed/Undirected Token Moving for Parameter b

In this section, we show that UNLABELED UNDIRECTED TOKEN MOVING and UNLABELED DIRECTED TOKEN MOVING are fixed-parameter tractable when parameterized by the number of allowed multi-slides b . Note that the case where $b \in \Omega(k)$ is already covered by our polynomial kernel result for k in Section 4.1.

Recall from Section 3.5 of the preliminaries that the graph $G_{\vec{M}}$ represents the induced subgraph $G[X]$, where X is an edge set that contains all the edges of the paths for the multi-slides in $\vec{M}$. Our algorithm for UNLABELED UNDIRECTED TOKEN MOVING is based

on the characterization of the graph $G_{\vec{M}}$ of a transformation sequence $\vec{M}$ of minimum size, where G is the graph of a kernel of maximum size $2k$ built according to the procedure in Lemma 18. In Lemmas 20 and 21, we show that there exists a transformation sequence $\vec{M}$ such that $G_{\vec{M}}$ is a forest of minimum-size Steiner trees with terminals in $(S \cup T) \setminus (S \cap T)$.

Given a graph G , a *Steiner tree* for a set of vertices $T \subseteq V(G)$ (called *terminals*) is a tree induced by a set of edges in G that includes all the vertices in T . The tree may include additional vertices from $V(G) \setminus T$ (called *Steiner vertices*). A *minimum Steiner tree* is one with the minimum number of edges (or minimum total weight in weighted graphs). The STEINER TREE problem takes as input a graph G , a set of terminals in G , and an integer κ and asks if G has a Steiner tree with at most κ edges. The problem has an algorithm that runs in $3^q \cdot n^{O(1)}$ time where q is the number of terminals [72].

To prove Lemma 20, we require the efficient (polynomial time algorithm) of Călinescu et al. [56] that allows us to obtain, given an undirected tree and configurations $S, T \subseteq V(G)$, a shortest transforming sequence from S to T .

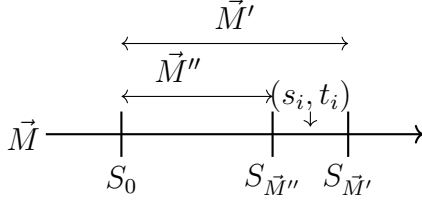
By considering all possible ways of partitioning vertices in $(S \cup T) \setminus (S \cap T)$ into Steiner trees, and counting the number of multi-slides required by each choice, in Theorem 22 we are able to obtain a fixed-parameter tractable algorithm for UNLABELED UNDIRECTED TOKEN MOVING when parameterized by b .

Hereafter, for an instance (G, S, T, b) of UNLABELED UNDIRECTED TOKEN MOVING, we refer to the kernel (G', S, T, b) built as per the procedure described in Lemma 18 as its *contracted instance*.

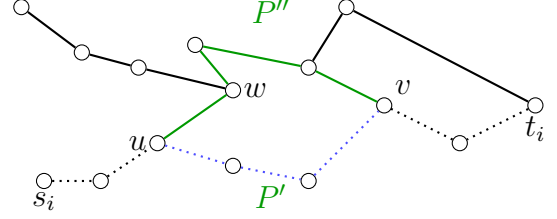
Lemma 20. *Let (G, S, T, b) be an instance of UNLABELED UNDIRECTED TOKEN MOVING and let (G', S, T, b) be its contracted instance. There exists a transformation sequence $\vec{M}$ that is a solution to (G', S, T, b) of minimum size such that $G'_{\vec{M}}$ is a forest.*

Proof of Lemma 20. Suppose to the contrary that each transformation sequence of minimum size induces (through the edges of the paths for its multi-slides) a subgraph containing a cycle. Of these, we consider the minimum size subsequence $\vec{M}'$ of a transformation sequence $\vec{M}$ consisting of multi-slides bracketing the formation of the first cycle in $G'_{\vec{M}}$: for the first cycle C formed, $\vec{M}'$ starts with the first multi-slide such that the path for the multi-slide contains a vertex in C , and ends with the first multi-slide that completes C . We use $\vec{M}''$ to refer to the sequence formed by removing the last multi-slide from $\vec{M}'$, and $m_i = (s_i, t_i)$ to refer to the last multi-slide in $\vec{M}'$.

In the graph $G'_{\vec{M}'}$, we use S_0 to denote the configuration before $\vec{M}'$ is executed, $S_{\vec{M}''}$ to denote the configuration after $\vec{M}''$ is executed, and $S_{\vec{M}'}$ to denote the configuration after



(a) Depiction of sequences by horizontal lines and timing of configurations by vertical lines.



(b) $G'_{\vec{M}'}$ with the dotted path for the multi-slide $m_i = (s_i, t_i)$; T' consists of all solid edges as well as the path from s_i to s' . No extra dotted edges are added to connect t , as it is already part of G_γ .

Figure 4.2: Sequences and $G_{\vec{M}'}$.

$\vec{M}'$ is executed. See Figure 4.2a. By definition, $S_{\vec{M}''}$ and $S_{\vec{M}'}$ differ only in the placement of a single token, which can be found on $s_i \in S_{\vec{M}''}$ and on $t_i \in S_{\vec{M}'}$.

Since m_i completes the cycle C , we can decompose C into two paths between a pair of vertices u and v , where u and v are both on the path for the multi-slide m_i . We use P'' to denote the path between u and v that is part of $G'_{\vec{M}''}$ and P' to denote the path between u and v that is part of the path for the multi-slide m_i . See Figure 4.2b.

Our goal is to form an alternative sequence of valid multi-slides such that P'' can be used instead of P' to reach configuration $S_{\vec{M}'}$. To make this possible, we identify the token on a vertex w on P'' that is in $S_{\vec{M}''}$ and is closest to s_i in $G'_{\vec{M}'}$. We will show that we can find a sequence of multi-slides $\vec{M}_2''$ that, when executed on G' starting from configuration S_0 , results in a configuration $S_{\vec{M}_2''}$ that differs from $S_{\vec{M}''}$ only in the placement of a single token, where $S_{\vec{M}_2''}$ has the vertex t_i and $S_{\vec{M}''}$ has the vertex w . We can then execute the multi-slide (s_i, b) to reach the configuration $S_{\vec{M}'}$ from $S_{\vec{M}_2''}$.

To obtain a contradiction and complete the proof, we need to show that $\vec{M}_2''$ does not use internal vertices of P' and that the size of $\vec{M}_2''$ is no greater than that of $\vec{M}''$. We form a tree T' that we initialize to be $G'_{\vec{M}''}$, and for each of s_i and t_i that is not in $G'_{\vec{M}''}$, we add to T' a subpath of the path for the multi-slide m_i with one endpoint in $G'_{\vec{M}''}$, and the other endpoint the vertex s_i or t_i . See Figure 4.2b. The fact that m_i closes the cycle C guarantees that if s_i and t_i are not in $G'_{\vec{M}''}$, such paths exist and that they contain no internal vertices of P' .

To compare the sizes of $\vec{M}''$ and $\vec{M}_2''$, we first observe that, by construction and the

definition of a multi-slide, every token in $V(G'_{\vec{M}''}) \cap S_0$ must be moved in order to transform S_0 into $S_{\vec{M}''}$. Since T' is a tree, we can invoke the polynomial-time algorithm of Călinescu et al. [56] on $(T', V(G'_{\vec{M}''}) \cap S_0, S_{\vec{M}''})$, allowing us to obtain a shortest transformation sequence of valid multi-slides from S_0 to $S_{\vec{M}''}$ in T' to form $\vec{M}''$. By Lemma 17, the number of multi-slides in $\vec{M}''$ cannot be greater than the number of tokens in $V(G'_{\vec{M}''}) \cap S_0$ and hence not greater than the number of multi-slides in $\vec{M}'$, as required to complete the proof. $\square$

Lemma 21. *For a contracted instance (G', S, T, b) of UNLABELED UNDIRECTED TOKEN MOVING with a transformation sequence $\vec{M}$ of minimum size such that $G'_{\vec{M}}$ is a forest, each tree in the forest is a minimum Steiner tree with terminals in $(S \cup T) \setminus (S \cap T)$ and Steiner vertices in the set of obstacles O , and such that each Steiner vertex in O is the source vertex of a multi-slide.*

Proof of Lemma 21. The fact that the terminals of the tree are in $(S \cup T) \setminus (S \cap T)$ follows from the minimality of the size of $\vec{M}$. Because each token in $G'_{\vec{M}}$ must be moved in a contracted instance, a minimum Steiner tree will contain the minimum number of vertices, and hence the minimum number of multi-slides. $\square$

Theorem 22. UNLABELED UNDIRECTED TOKEN MOVING is fixed-parameter tractable when parameterized by b .

Proof of Theorem 22. Given an instance (G, S, T, b) of UNLABELED UNDIRECTED TOKEN MOVING, we first form in polynomial time a contracted instance (G', S, T, b) according to the procedure described in Lemma 18. Then we produce all possible partitions of vertices in $(S \cup T) \setminus (S \cap T)$ into at most b Steiner trees, starting first with a single tree, then two, and so on.

When considering the use of t trees, we first consider all possible ways of partitioning the vertices of $(S \cup T) \setminus (S \cap T)$ into t groups, where each group has an equal number of vertices in $S \setminus T$ and $T \setminus S$, and then run the Dreyfus–Wagner Steiner tree algorithm [72] on each such group of vertices. Because in each Steiner tree, each token must multi-slide and does so exactly once according to Lemma 17, the number of multi-slides associated with each Steiner tree T_{steiner} will be $|(S \setminus T) \cap V(T_{\text{steiner}})| + |O \cap V(T_{\text{steiner}})|$ where O is the set of obstacles. If the total number of multi-slides in a forest is at most b , we return YES. If we have not returned YES using t trees, we consider $t + 1$ trees. If the procedure does not return YES after considering all values of t , we return NO.

The correctness of the algorithm follows from Lemmas 18, 20, and 21. Let $r = |(S \cup T) \setminus (S \cap T)| = |S \Delta T|$. Any solution using at most b multi-slides must move all tokens in

$S \setminus T$, hence $|S \setminus T| \leq b$ and therefore $r = 2|S \setminus T| \leq 2b$. For each $t \in \{1, \dots, \min\{b, r\}\}$ we enumerate all partitions of the r terminals into t (balanced) groups and, for each group of q terminals, compute a minimum Steiner tree using the Dreyfus–Wagner algorithm in time $O(3^q \cdot n^{\mathcal{O}(1)})$. Summed over a fixed partition, the time to compute the Steiner trees is $O(3^r \cdot n^{\mathcal{O}(1)})$. The number of partitions is at most $2^{O(r \log r)}$. Hence, the overall running time is $2^{\mathcal{O}(b \log b)} \cdot n^{\mathcal{O}(1)}$ which is fixed-parameter tractable with respect to b . $\square$

Like in the case of undirected graphs, our algorithm for the directed case relies on the characterization of the graph $D_{\vec{M}}$ of a transformation sequence $\vec{M}$ of minimum size, where D is the graph of a kernel of maximum size $2k$ built according to the procedure in Lemma 18. Recall from Section 3.5 of the preliminaries that the digraph $D_{\vec{M}}$ represents the induced subgraph $D[X]$ where X is an arc set that contains all the arcs of the directed paths for the multi-slides in $\vec{M}$. Furthermore, a *clearing multi-slide* is one whose source vertex is an obstacle vertex in O and a *filling multi-slide* is one whose target vertex is an obstacle vertex in O . We show, in Lemma 27, that for any yes-instance there exists a transformation sequence $\vec{M}$ such that $D_{\vec{M}}$ is a directed forest. As a replacement for the Steiner tree approach, the fact that we can bound the size of $D_{\vec{M}}$ suggests the use of color coding machinery to determine whether D contains a labeled subgraph of the correct form to be $D_{\vec{M}}$. Unfortunately, this machinery (particularly, Theorem 23) cannot be used directly; in Theorem 28 we make small modifications to use it.

Theorem 23 ([7]). *Let H be a directed forest on h vertices. Let D be a directed graph. A copy of H in D , if it exists, can be found in $2^{\mathcal{O}(h)} \cdot n^2 \cdot \log n$ worst-case time. Moreover, if a weight function $\omega : A(D) \rightarrow \mathbb{R}$ is defined on the arcs of D , then the algorithm can be adapted to find the copy of H in D with the maximum total weight.*

After removing extraneous vertices from D in Lemma 24, we demonstrate that $D_{\vec{M}}$ forms a directed forest in Lemma 27; to show that we can ignore cycles, we focus on a minimal digraph whose underlying undirected graph contains a cycle as a counterexample. Formally, we call a digraph D a *circle digraph* if D is connected and the vertices in $V(D)$ can be partitioned into *cycle vertices*, forming a cycle C in the underlying undirected graph of D and *forest vertices*, forming a directed forest of directed trees such that each tree has its *root vertex* adjacent to a cycle vertex and all its arcs are directed toward the root or all arcs are directed away from the root.

Hereafter, for an instance (D, S, T, b) of UNLABELED DIRECTED TOKEN MOVING, we refer to the kernel (D', S, T, b) built as per the procedure described in Lemma 18 as its *contracted instance*. An instance (D, S, T, b) of UNLABELED DIRECTED TOKEN MOVING is said to be a *contracted circle instance* whenever D is a circle digraph and $S \cup T = V(D)$.

Lemma 24. *Let (D, S, T, b) be an instance of UNLABELED DIRECTED TOKEN MOVING, (D', S, T, b) be its contracted instance and $v \in S \cap T$. (D', S, T, b) is a yes-instance if and only if $(D' - \{v\}, S \setminus \{v\}, T \setminus \{v\}, b)$ is a yes-instance when any of the following conditions hold:*

- *There is no directed path from any vertex in $S \setminus T$ to v .*
- *There is no directed path from v to any vertex in $T \setminus S$.*
- *Every directed path from any vertex in $S \setminus T$ to v contains at least $b + 1$ obstacles, and every directed path from v to any vertex in $T \setminus S$ contains at least $b + 1$ obstacles.*

Proof of Lemma 24. By Lemma 17, in any shortest transformation sequence, v is the source of a clearing multi-slide and the target of a filling multi-slide, or its token blocks the multi-slide of another token. Because each pair of clearing and filling multi-slides forms a directed path from a vertex in $S \setminus T$ to a vertex in $T \setminus S$ (as does any multi-slide whose source vertex is in $S \setminus T$ and target vertex is in $T \setminus S$), under the first two conditions v can neither be part of such a multi-slide nor block such a multi-slide.

For the third condition, we observe that b multi-slides are not sufficient either to move enough other tokens initially on obstacle vertices for there to be a clear directed path between v and a vertex in $T \setminus S$, nor for the token initially on v to be the only token blocking a directed path. $\square$

Lemma 25. *If there exist instances (D, S, T, b) of UNLABELED DIRECTED TOKEN MOVING for which every transformation sequence $\vec{M}$ of minimum size yields a $D_{\vec{M}}$ that is not a directed forest, then at least one of those instances must be a contracted circle instance.*

Proof of Lemma 25. Of all those instances, let us consider an instance (D, S, T, b) where $|V(D)|$ is minimized, $|A(D)|$ is minimized, and the size of a shortest transformation sequence $\vec{M}$ is minimized (in that order). We call such an instance a *minimal instance*.

First, note that we can assume that each vertex in $V(D)$ is in S or T , since applying the procedure described in the proof of Lemma 18 would contradict the minimality of $|V(D)|$.

We show that in a minimal instance, the last multi-slide in the shortest transformation sequence $\vec{M}$ is the one that forms one or more cycles in the underlying undirected graph of $D_{\vec{M}}$. Suppose in contrast that the first cycle is formed by a multi-slide m_i that is not the last in the sequence. We then define S_{m_i} to be the configuration after m_i is executed. The instance $(D_{\vec{M}}, S, S_{m_i}, b)$ satisfies the statement of the lemma and has a minimum size

transformation sequence consisting of at least one fewer multi-slides, contradicting the minimality of the minimal instance.

We now show that $D_{\vec{M}}$ cannot contain more than one connected component nor have an underlying undirected graph that contains more than one cycle. If instead $D_{\vec{M}}$ contains more than one connected component, since the multi-slides in each component are independent of each other, the multi-slides that form the components that do not form the cycle could have taken place after the last multi-slide in $\vec{M}$, contradicting the minimality of the size of $\vec{M}$. If the last multi-slide $m_\ell = (s_\ell, t_\ell)$ closes more than one cycle in the underlying undirected graph, then we let $\vec{M}'$ be the sequence obtained by removing m_ℓ from $\vec{M}$ and P_{m_ℓ} be the path for the multi-slide m_ℓ . We let v_0, v_1 , and v_2 be the first, second, and third intersections of P_{m_ℓ} with $D_{\vec{M}'}$ such that $(v_0, v_1) \notin A(D_{\vec{M}'})$ and $(v_1, v_2) \notin A(D_{\vec{M}'})$, respectively. Such vertices must exist since m_ℓ is assumed to close more than one cycle in the underlying undirected graph. We note that we could have longer directed paths from v_0 to v_1 or from v_1 to v_2 but the same argument still applies. We create a new instance by deleting all arcs added by m_ℓ from v_0 to v_1 and all arcs added by m_ℓ from v_1 to v_2 and instead adding an arc from v_0 to v_2 ; since the directed path from v_0 to v_2 has at least three arcs, we have found a smaller instance that uses fewer arcs, contradicting the minimality of $|A(D)|$. Hence, $D_{\vec{M}}$ consists of a single component whose underlying undirected graph contains a single cycle.

Finally, due to the minimality of the instance, we can assume that each vertex is a source or a target vertex for a multi-slide. Moreover, due to the minimality of $\vec{M}$, $\vec{M}$ cannot contain any multi-slide that does not pass through C . Consequently, we conclude that any vertex outside of C must be connected by one directed path to a vertex in C or connected by one directed path from a vertex in C , forming a contracted circle instance (see Figure 4.3). $\square$

We use Lemma 25 in the proof of Lemma 27, where we use the structure of a circle digraph to form a transformation sequence that does not form a cycle. We number the cycle vertices of the circle digraph as $v_1, v_2, \dots, v_\kappa$ in clockwise order, and observe that for each i , either (v_i, v_{i+1}) or (v_{i+1}, v_i) is an arc in the contracted circle instance graph. We say that a subsequence of vertices $v_a, v_{a+1}, \dots, v_b$ is a *forward cycle segment* if (v_i, v_{i+1}) is an arc for each $a \leq i < b$ and a *backward cycle segment* if (v_{i+1}, v_i) is an arc for each $a \leq i < b$. Since the paths for multi-slides are directed paths, no multi-slide can use arcs in more than one cycle segment.

Of particular interest are the *junction vertices* shared by two consecutive cycle segments, where each is a *source* or a *sink* in both cycle segments. We refer to the *in-pool* of a source junction vertex v as the set containing v and any tree vertex that can reach v by a directed

path, and the *out-pool* of a sink junction vertex v as the set containing v and any tree vertex that can be reached by a directed path from v . Since each cycle segment starts and ends with a junction vertex, we can refer without ambiguity to the in-pool and out-pool of a cycle segment. By partitioning the multi-slides by cycle segments, we form a sequence of directed trees, thereby allowing us to apply Lemma 26.

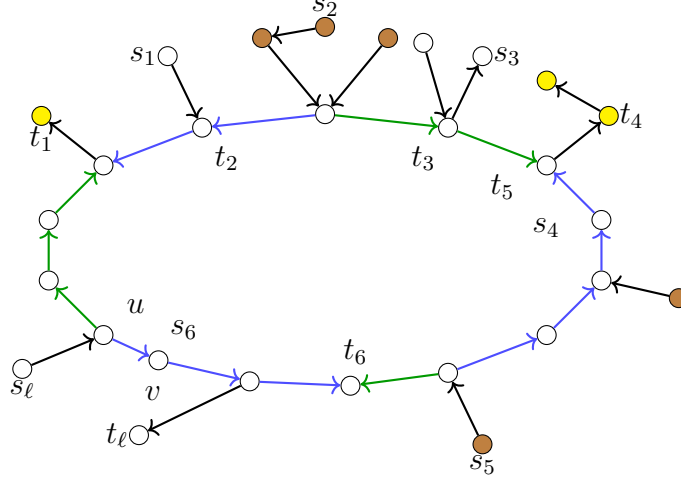


Figure 4.3: Example circle graph with cycle segments highlighted, in-pools marked in brown, and out-pools marked in yellow, and sources and sinks of linking moves shown.

Lemma 26. *Given a directed tree D , two configurations S and T of D such that every leaf of D is in $(S \cup T) \setminus (S \cap T)$, and a one-to-one mapping μ from S to T such that there is a directed path from each $s \in S$ to $\mu(s) \in T$ (and $s \neq \mu(s)$ for all s), there exists a transformation sequence from S to T in D .*

Proof of Lemma 26. We proceed by induction on the size of the directed tree D , observing that if $|V(D)| = 2$ then the statement clearly holds.

If there exists a leaf $v \in T \setminus S$, we let $u = \mu^{-1}(v)$ and let P_{uv} be the unique directed path from u to v . We let w be the closest vertex to v on P_{uv} such that $w \in S$. Note that w could be equal to u . Moreover, all the vertices in P_{wv} are in $T \setminus S$. Then, we form a smaller instance D' and configurations S' and T' as follows. If $w \neq u$ then we delete v from D (and T), remove w from S , and set $\mu(u) = \mu(w)$; since we have a directed path from u to w and a directed path from w to $\mu(w)$, we have a directed path from u to $\mu(w)$. If $w = u$ then we simply delete v from S and if w becomes a leaf, we also delete w . When w does not become a leaf, it becomes a free vertex (not in $S \cup T$). To form a transformation

sequence from S to T in D , we start with the move (w, v) and then add the transformation sequence for the smaller instance.

Now we consider the situation in which all the leaves of D are in $S \setminus T$. For a leaf $u \in S \setminus T$, we let $v = \mu(u)$ and, again, we let P_{uv} be the unique directed path from u to v . We find the closest vertex w to v on P_{uv} such that $w \in T$. All the vertices in P_{uv} are in $S \setminus T$. We let $\mu(w) = x$ (assuming $w \in S \cap T$) and $\mu^{-1}(w) = y$. We can form a smaller instance by removing u , and form a transformation sequence from the transformation sequence of the smaller instance followed by the move (u, w) . To do so, we set $\mu(y) = x$ and $\mu(w) = v$; by construction, there is a directed path from w to v , from w to x , and from y to w . If $w \in T \setminus S$ then we let $\mu^{-1}(w) = y$ and form the smaller instance in a similar manner but setting $\mu(y) = v$. $\square$

Lemma 27. *If a contracted instance (D, S, T, b) is a yes-instance of UNLABELED DIRECTED TOKEN MOVING, then there exists a transformation sequence $\vec{M}$ of minimum size such that $D_{\vec{M}}$ is a directed forest.*

Proof of Lemma 27. For the sake of contradiction, we assume that (D, S, T, ℓ) is an instance for which every transformation sequence $\vec{M}$ of minimum size yields a digraph $D_{\vec{M}}$ that contains a cycle in the underlying undirected graph. By Lemma 25, we can assume that (D, S, T, ℓ) is a contracted circle instance, $m_\ell = (s_\ell, t_\ell)$ is the last multi-slide in $\vec{M}$, $D_{\vec{M}}$ contains a single cycle in the underlying undirected graph, $\vec{M}'$ is formed by removing m_ℓ from $\vec{M}$, and $D_{\vec{M}'}$ is a directed forest.

As in the proof of Lemma 20, we decompose the cycle into two undirected paths P and P' in the underlying undirected graph, both between vertices u and v , where u and v are both on the path for the multi-slide m_ℓ . We focus on $D_{\vec{M}} - E(P')$ where P' is the path added by the multi-slide m_ℓ .

We construct a sequence $\vec{M}_2$ of the same size as $\vec{M}$ that achieves the same transformation, but does not use P' . By Lemma 26, it suffices to construct a mapping $\mu : S \rightarrow T$ such that there exists a directed path from each $s \in S$ to $\mu(s) \in T$.

We break P into cycle segments. Starting at the cycle segment containing u and s_ℓ , we can set $\mu(s_\ell)$ to be a target vertex w in T in the out-pool of the same segment and whose source vertex is a vertex of another segment or another segment pool. We keep all other multi-slides of the same segment intact. Note that since there is an arc that goes in the opposite direction and is incoming into the junction out-pool vertex of the segment, w must exist. In the next cycle segment, we start by mapping the source vertex s_w whose target is w to the target vertex of any multi-slide whose path passes through s_w , and

do so for every source vertex whose target is mapped until we get to one source vertex in the segment's in-pool with no multi-slide whose path passes through the vertex. We map this source vertex to a target vertex w' in T in the out-pool of the next unconsidered segment and whose source vertex is a vertex of yet another segment or yet another segment pool. Repeating this until the last cycle segment of P produces the required mapping of Lemma 26. $\square$

To check if a contracted instance (D, S, T, b) is a yes-instance, it suffices to determine whether a labeled version of D contains a subgraph of the correct form to be $D_{\vec{M}}$ for some $\vec{M}$ of minimum size. When we assign labels to the vertices of D such that the vertices of $S \setminus T = j$ are labeled from s_1 to s_j , the vertices of $T \setminus S = j$ are labeled from t_1 to t_j , and all other vertices are assigned the label j , we say that D is $(2|S \setminus T| + 1)$ -labeled. We use $\text{lab}(v)$ to denote the label of vertex $v \in V(D)$ and we have $2|S \setminus T| + 1$ distinct labels.

When (D, S, T, b) is a contracted yes-instance with $\vec{M}$ being a transformation sequence of minimum size, $D_{\vec{M}}$ is a directed forest and has at most $b + |S \setminus T| \leq 2b$ vertices, as otherwise $b + 1$ multi-slides are required. Thus, we can enumerate all possible $(2|S \setminus T| + 1)$ -labeled directed forests of size at most $2b$ and check whether any one of them implies a yes-instance and can be found as a subgraph of D .

For a $(2|S \setminus T| + 1)$ -labeled directed forest H , we let $S' = \{v \in V(H) \mid \text{lab}(v) \in \{s_1, \dots, s_j, j\}\}$ and $T' = \{v \in V(H) \mid \text{lab}(v) \in \{t_1, \dots, t_j, j\}\}$. Then H is said to be a *witness* for (D, S, T, b) if (H, S', T', b) is a yes-instance and there is a copy of H in D such that the labeling of the vertices in $(S \cup T) \setminus (S \cap T)$ is respected.

Theorem 28. UNLABELED DIRECTED TOKEN MOVING is *fixed-parameter tractable* when parameterized by b .

Proof of Lemma 28. For any instance of UNLABELED DIRECTED TOKEN MOVING, we can transform the instance using Lemmas 18 and 24 into a contracted instance (D, S, T, b) such that for every vertex $v \in S \cup T$ (that is, every vertex v in D), v is on a directed path from a vertex in $S \setminus T$ to a vertex in $T \setminus S$.

By Lemma 27, we know that if (D, S, T, b) is a yes-instance, then there exists a transformation sequence $\vec{M}$ of minimum size such that $D_{\vec{M}}$ is a directed forest. Additionally, at most $b - |S \setminus T|$ multi-slides can be of tokens on vertices not in $S \setminus T$. Hence, the total number of vertices in $D_{\vec{M}}$ is at least $2|S \setminus T|$ and at most $b + |S \setminus T|$.

To check if (D, S, T, b) is a yes-instance, we make D $(2|S \setminus T| + 1)$ -labeled and search for a witness by enumerating all $(2|S \setminus T| + 1)$ -labeled directed forests on κ vertices, for each

value greater than or equal to $2|S \setminus T|$ and smaller than or equal to $b + |S \setminus T|$. We observe that the total number of such forests is bounded by a function of b . For each directed forest H , we determine whether H is a witness by determining whether (H, S', T', b) is a yes-instance (which can be achieved in fixed-parameter tractable running time with respect to parameter b by brute force) and whether there is a subgraph of D that is a copy of H and that respects the labels.

For the latter, we create edge-weighted digraphs D' and H' with vertices that correspond to those in D and H . We assign all edges in D' weight one. Each vertex in D' or H' corresponding to a vertex with label s_i is given i degree-one in-neighbors, connected by arcs with weight iq , and each vertex in D' or H' corresponding to a vertex with label t_j is given $|S \setminus T| + j$ degree-one out-neighbors, connected by arcs with weight $(|S \setminus T| + j)q$.

We can now invoke Theorem 23 on D' and H' . If H has a copy in D , then the copy of H' in D' with the maximum total weight on its arcs will include all arcs of weight greater than one, and thus we must find a label-preserving copy. If we find a witness, we return YES. Otherwise, we return NO.

The algorithm enumerates all $(2|S \setminus T| + 1)$ -labeled directed forests of size at most $2b$. The number of such labeled forests is at most $2^{O(b \log b)}$. For each forest H in this collection, the algorithm verifies whether (H, S', T', b) is feasible (in time $2^{O(b)}$) and then checks whether H occurs as a label-respecting subgraph of D using the color-coding algorithm of Alon et al. [7], which runs in $2^{O(b)} n^2 \log n$ time. Thus, the total running time of the algorithm is $2^{O(b \log b)} n^{O(1)}$. Thus, UNLABELED DIRECTED TOKEN MOVING is fixed-parameter tractable with respect to b . $\square$

4.3 $W[1]$ -hardness of Labeled Undirected/Directed Token Moving for parameter b

In this section, we show that unlike their counterparts where the tokens are unlabeled, LABELED UNDIRECTED TOKEN MOVING and LABELED DIRECTED TOKEN MOVING are $W[1]$ -hard when parameterized by b or even $b + f$ on general graphs where f denotes the number of token-less vertices outside $S \cup T$. We give reductions from the MULTICOLORED SUBGRAPH ISOMORPHISM problem.

We say that two graphs G and H are *isomorphic* if there exists a bijection $\mu : V(G) \rightarrow V(H)$ such that $\{u, v\} \in E(G)$ if and only if $\{\mu(u), \mu(v)\} \in E(H)$. When dealing with vertex-colored graphs $(G, \mathcal{C}_G)$ and $(K, \mathcal{C}_K)$, we additionally require that the isomorphism

preserves colors: $\mathcal{C}_G(u) = \mathcal{C}_H(\mu(u))$ for all $u \in V(G)$. The MULTICOLORED SUBGRAPH ISOMORPHISM problem asks whether a given vertex-colored graph $(G, \mathcal{C}_G)$ where each vertex has one color contains a subgraph that is isomorphic to another given vertex-colored graph $(K, \mathcal{C}_K)$ in which every vertex has one distinct color, while preserving color constraints. See Figure 4.4. This problem is $W[1]$ -hard when parameterized by $|V(K)|$ [142], even when K is restricted to being a connected bipartite graph in which every vertex is of degree three.

For clarity, we refer to nodes in the reduced instance and vertices in the MULTICOLORED SUBGRAPH ISOMORPHISM instance. Both of our reductions create a *vertex node* for w for each $w \in V(G)$ (of color present in $\mathcal{C}_K$) and then use the structure of K to group these nodes by color to form *supernodes*. By judicious assignment of labels to tokens that must multi-slide between supernodes and the tight budget, we ensure that reconfiguration can occur only if we can clear tokens on a set of vertex nodes that form a subgraph of G isomorphic to K . We start with the directed case and then explain the necessary modifications for the undirected case.

Labeled Directed Token Moving

We explain the reduction in three steps: forming a directed acyclic graph (DAG) K' from K that provides the structure of D (the digraph of the LABELED DIRECTED TOKEN MOVING instance), creating and connecting supernodes, and adding gadgets to restrict token movement.

Step 1. To form a directed acyclic graph K' , we create a vertex $k_i \in V(K')$ for each vertex $i \in V(K)$, making use of [breadth-first search](#) to assign each vertex to a level based on the distance from an arbitrary vertex indexed 1 that is at level 0. Note that each remaining vertex k_i is assigned to the minimum level ℓ such that there is a path of $\ell - 1$ vertices from 1 to i in K . We form a directed acyclic graph (Figure 4.4c), by adding an arc in K' between a vertex in level ℓ and another in level $\ell + 1$ when their corresponding vertices are adjacent in K . Note that these arcs represent all edges in K .

Step 2. We can safely ignore any edge in G between vertices of the same color or between colors not connected by an edge in K , since no such edge can be part of a subgraph of G isomorphic to K . For each color C_i (for $i \in V(K)$), *supernode* D_i consists of all vertex nodes for vertices in C_i in G ; we consider the supernode D_i to be at the same level as k_i in K' . To form D , we add an arc between any vertex node in D_i and a vertex node in

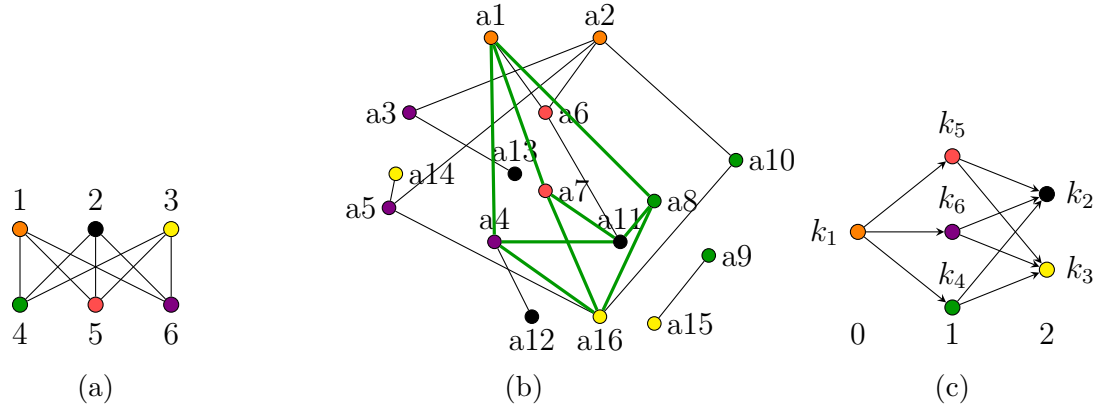


Figure 4.4: (a) K (b) G (c) K' showing level numbers.

D_j when (k_i, k_j) is an arc in K' and the vertices represented by those vertex nodes are adjacent in G . See Figure 4.5a. When there exists an arc from a vertex in D_i to a vertex in D_j , we say that D_i is a *super-in-neighbor* of D_j and that D_j is a *super-out-neighbor* of D_i .

Step 3. To ensure that each edge of K is present, we associate a token with each arc in K' . For arc (k_i, k_j) , in S , the associated token is placed on a node in the *source gadget* of D_i , and in T , the associated token is placed on a node in the *target gadget* of D_j , where the source and target gadgets will be defined shortly. Accordingly, for each supernode, there is one node in its source gadget for each super-out-neighbor, and there is one node in its target gadget for each super-in-neighbor. For each token in $S \setminus T$, its source node and target node are in the source gadget and the target gadget of consecutive levels.

For LABELED DIRECTED TOKEN MOVING, the source gadget of a supernode consists of a set of nodes with tokens connected by arcs into each vertex node in the supernode and the target gadget consists of a set of nodes without tokens connected by arcs from each vertex node in the supernode. See Figure 4.5b. In addition, associated with each supernode is a *storage gadget* consisting of a single node that is the in-neighbor and out-neighbor of every vertex node in the supernode. As their names suggest, the union of nodes in the source gadgets equals $S \setminus T$ and the union of nodes in the target gadgets equals $T \setminus S$. We also set $S \cap T$ to all the vertex nodes. By construction, $|S \setminus T| = |E(K)|$ and $f = |V(K)|$; we set $b = |E(K)| + 2|V(K)|$. The latter and the fact that the reduction runs in polynomial time ensures that we obtain a valid FPT-reduction.

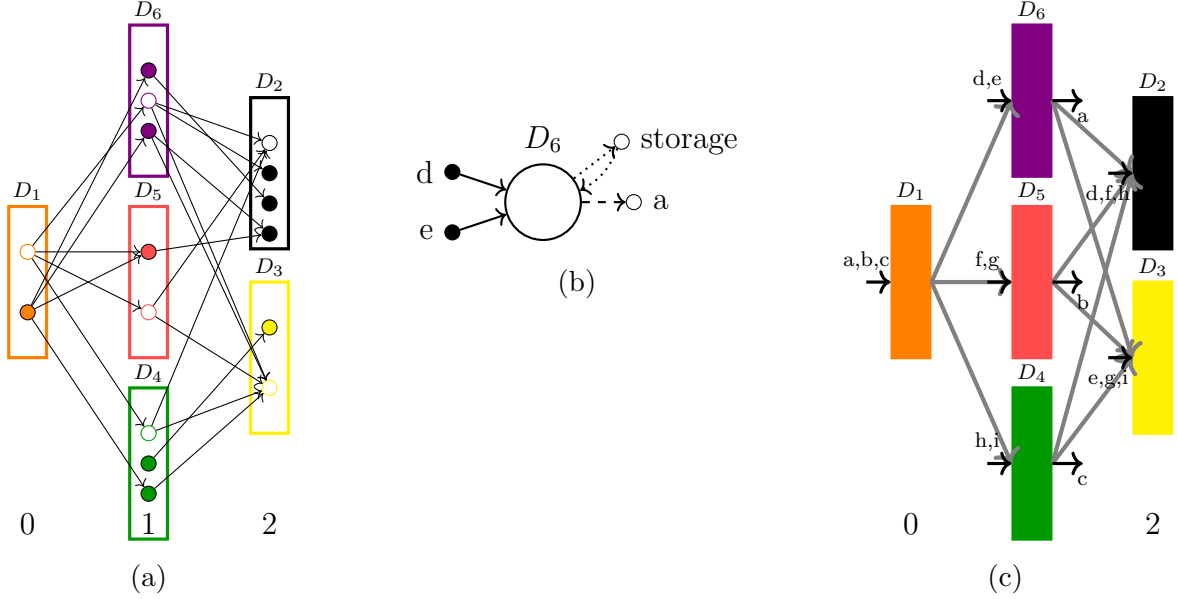


Figure 4.5: (a) Supernodes of D with vertex nodes corresponding to a subgraph isomorphic to K' shown as unfilled circles. (b) Source, target, and storage gadgets for D_6 with edges shown using solid, dashed, and dotted arcs, respectively. Two tokens are initially present in the source gadget for D_6 (labeled d and e), and one node is in the target gadget and must receive the token with label a . (c) A schematic diagram showing supernodes as rectangles and source gadgets and target gadgets as labeled incoming and outgoing black arrows. For example, the source gadget attached to D_6 has two source gadget nodes whose tokens have target nodes in out-neighbors D_2 and D_3 , and the target gadget attached to D_6 has one node, the target node of a token in in-neighbor D_1 .

Lemma 29. *If $((G, \mathcal{C}_G), (K, \mathcal{C}_K))$ is a yes-instance of MULTICOLORED SUBGRAPH ISOMORPHISM, then (D, S, T, b) is a yes-instance of LABELED DIRECTED TOKEN MOVING.*

Proof of Lemma 29. We will specify a sequence of b multi-slides, consisting of c clearing multi-slides, $S \setminus T$ multi-slides of tokens on nodes in $S \setminus T$ to nodes in $T \setminus S$, and finally c filling multi-slides.

We define VK as the set of vertices of G in the subgraph isomorphic to K . For our clearing multi-slides, we multi-slide the tokens on the vertex node for w for each $w \in VK$ to adjacent storage gadgets. Since each vertex $w \in VK$ has a different color, each vertex node for w is in a different supernode, and hence the storage gadgets are all distinct.

Moreover, for each arc (k_i, k_j) in K' , we have cleared the endpoints of an arc from D_i to D_j in D , allowing all the multi-slides from $S \setminus T$ to $T \setminus S$ to take place. Finally, the c tokens that we moved in the clearing multi-slides can be returned to their original nodes. $\square$

Lemma 30. *If (D, S, T, b) is a yes-instance of LABELED DIRECTED TOKEN MOVING, then $((G, \mathcal{C}_G), (K, \mathcal{C}_K))$ is a yes-instance of MULTICOLORED SUBGRAPH ISOMORPHISM.*

Proof of Lemma 30. Due to the sizes of $S \setminus T$ and b , the transformation sequence consists of c clearing multi-slides, $b - 2c$ multi-slides of tokens on nodes in $S \setminus T$ to nodes in $T \setminus S$, and c filling multi-slides, although the order among them is unconstrained. As a consequence of the construction, we observe that before the multi-slide of a token from a source gadget of D_i to a target gadget of D_j , there must be two clearing multi-slides, one of a vertex node for u in D_i and one of a vertex node for v in D_j such that $(u, v) \in E(G)$.

To be able to clear enough nodes to allow all $S \setminus T$ multi-slides of tokens from source gadgets to target gadgets, we must be able to select c vertex nodes to move such that for every edge (k_i, k_j) in K' , there is an edge $(u, v) \in E(G)$ such that $u \in C_i$ and $v \in C_j$, and the vertex nodes for u and v have both been cleared. By definition, the selected vertex nodes correspond to the vertices in a subgraph of G isomorphic to K' , and hence to K , and that preserves color constraints. This proves that $((G, \mathcal{C}_G), (K, \mathcal{C}_K))$ is a yes-instance of the MULTICOLORED SUBGRAPH ISOMORPHISM problem. $\square$

Combining Lemmas 29 and 30 with the fact that $b + f = \mathcal{O}(|V(K)| + |E(K)|) = \mathcal{O}(c)$, we obtain the following theorem.

Theorem 31. LABELED DIRECTED TOKEN MOVING is $W[1]$ -hard when parameterized by $b + f$.

Labeled Undirected Token Moving

We use the same basic structure as in the previous reduction, but need extra machinery to ensure that if a token multi-slides from D_i to D_j in the undirected graph D constructed by the reduction, it cannot find a route that avoids all edges corresponding to edges in G with endpoints of colors C_i and C_j . To this end, we introduce *superedges* and a *clock gadget*, defined below, and for each arc $(k_i, k_j) \in E(K')$, specify the numbers of tokens to move from D_i to D_j and D_j to D_i based on the levels of k_i and k_j . Source gadgets, target gadgets, and storage gadgets all consist of nodes connected by a single undirected edge to all vertex nodes in a supernode.

To construct G for LABELED UNDIRECTED TOKEN MOVING, we construct supernodes in the same way as in D , and although H is undirected, use the terms level, super-in-neighbor, and super-out-neighbor based on the structure of K' . In addition, we form an *edge path* $p\{x, y\}$ of length Q (to be defined later) for each edge $xy \in E(G)$, and for each arc (k_i, k_j) in K' , the *superedge* $D_{i,j}$ consisting of all edge-paths $p\{x, y\}$ such that $x \in C_i$ and $y \in C_j$. We consider the level of $D_{i,j}$ to be the same as the level of D_i . To connect the vertex nodes and edge paths in H , for each superedge $D_{i,j}$ and each edge path $p(x, y)$ in $D_{i,j}$, we add an edge from the vertex node for x to one endpoint of the edge path $p(x, y)$ and an edge from the other endpoint of $p(x, y)$ to the vertex node for y .

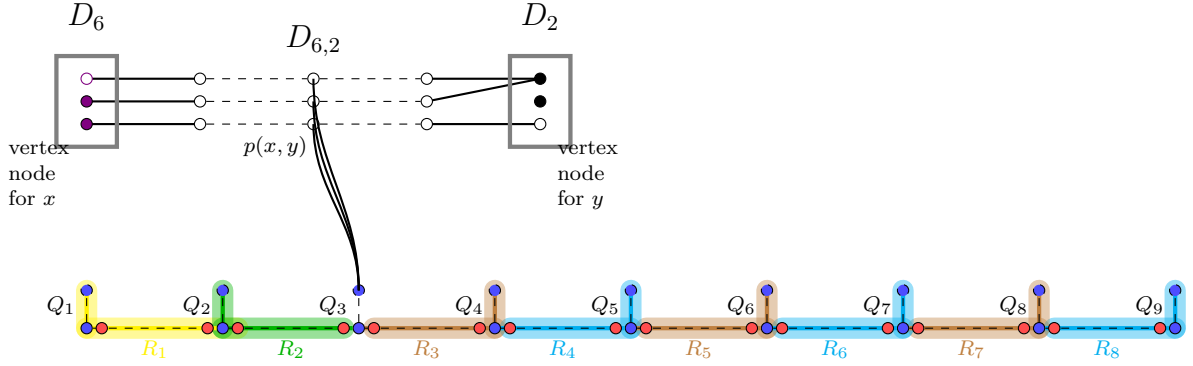


Figure 4.6: Edge-paths for superedge $D_{6,2}$ connected to Q_3 in the clock gadget (dashed lines represent long paths). Red nodes represent the left and right end of a roping path and the blue nodes represent the top and bottom ends of a storage path. Regions highlighted in yellow, green, brown, and blue show where tokens currently are. The first two clock segments have moved their tokens to target vertices, thus the clock is at position 3.

For convenience, we refer to the tokens in the source gadget attached to D_i destined for the target gadget attached to D_j as the D_i - D_j tokens. Our proofs hinge on showing that the D_i - D_j multi-slides (for the D_i - D_j tokens) pass through superedge $D_{i,j}$. To do so, for each superedge $D_{i,j}$ at level ℓ , we create r_ℓ D_i - D_j tokens (on nodes in the source gadget of D_i and whose targets are nodes in the target gadget of D_j) and r_ℓ D_j - D_i tokens, where for the last level z , $r_z = L = 3c^2/2$ and for any other level ℓ , $r_\ell = L \cdot r_{\ell+1}$.

The *clock gadget* is designed to allow the freeing of one edge path (all of which are full of tokens) per superedge at a time in increasing order of level, which we call the *clock numbering* of the superedges (that is, the superedges in one level are numbered in order before the superedges of the next level). See Figure 4.6. For large values Q and R to be defined later, the clock gadget consists of $|E(K)|$ storage paths $Q_1, \dots, Q_{|E(K)|}$, each of

length Q , and $|E(K)| - 1$ *roping paths* $R_1, \dots, R_{|E(K)|-1}$, each of length R . Referring to the two endpoints of each storage path as the *top end* and the *bottom end* of the storage path and the two endpoints of each roping path as the *left end* and the *right end* of the roping path, for each i , we add edges connecting the bottom end of Q_i to the right end of R_{i-1} (if it exists) and to the left end of R_i (if it exists).

We view tokens initially on the clock gadget as being grouped into $|E(K)| - 1$ *clock segments* (where clock segment i is formed of R_i and Q_{i+1}), each containing $Q + R$ tokens, such that Q_1 is token-free before the multi-slides of the tokens in the first clock segment from source to target nodes, Q_2 is token-free between the multi-slides of the tokens in the first and second clock segments, and $Q_{|E(K)|}$ is token-free only after the multi-slides of all tokens in all clock segments. All nodes in Q_1 are in $T \setminus S$, all nodes in $Q_{|E(K)|}$ are in $S \setminus T$, and for a clock segment i , the tokens are on R_i and Q_{i+1} , in order from left to right and bottom to top, and must end up on the nodes of Q_i and R_i , in order from top to bottom and from left to right. We say that the *clock is at position p* whenever the top end of Q_p does not have a token. Finally, we connect the top end of each Q_i to the middle node in each edge path in the superedge clock numbered i .

To complete the construction, we set S to consist of nodes in the source gadgets, source nodes in the clock gadget, and all vertex nodes and edge paths nodes, and T to consist of nodes in the target gadgets, target nodes in the clock gadget, and all vertex nodes and edge paths nodes. The number of free nodes in H is then the total size of the storage gadgets, or $f = |V(K)|$.

We now choose sufficiently large values of Q and R to control token movement. We set $Q = 2L^* + |V(K)| + 1$, where L^* is the total number of nodes in the source gadgets of all supernodes. For the number of multi-slides b , we sum $(|E(K)| - 1)(Q + R)$ multi-slides for the $|E(K)| - 1$ clock segments, L^* multi-slides of tokens from source to target gadgets, $2c$ clearing and filling multi-slides for c vertex nodes, and $2Q|E(K)|$ clearing and filling multi-slides for one edge path per superedge. Thus, we set $b = (|E(K)| - 1)(Q + R) + L^* + 2|V(K)| + 2Q|E(K)|$. We let $R = (|E(K)| - 1)Q + L^* + 2|V(K)| + 2Q|E(K)| + 1$, so that, written in a more convenient form, we have $b = |E(K)|R - 1$. The latter and the fact that the reduction runs in polynomial time ensures that we obtain a valid FPT-reduction.

Lemma 32. *If $((G, \mathcal{C}_G), (K, \mathcal{C}_K))$ is a yes-instance of MULTICOLORED SUBGRAPH ISOMORPHISM, then (H, S, T, b) is a yes-instance of LABELED UNDIRECTED TOKEN MOVING.*

Proof of Lemma 32. We specify a sequence of multi-slides that clears one vertex node in each supernode, completes all the multi-slides between $S \setminus T$ and $T \setminus S$, and then fills the

cleared vertex nodes. As in the proof of Lemma 29, we define VK as the set of nodes of G in the subgraph isomorphic to K , and to clear vertex nodes, we move the tokens on the vertex node for w for each $w \in VK$ to adjacent storage gadgets.

Clearing vertex nodes is not sufficient; the D_i - D_j and the D_j - D_i multi-slides are still blocked by tokens in superedge $D_{i,j}$. For each superedge $D_{i,j}$ clock numbered p , we clear the edge path corresponding to the edge in VK between the vertices of colors C_i and C_j by moving its tokens into Q_p , executing the D_i - D_j and the D_j - D_i multi-slides, returning the tokens from Q_p to the edge path, and then moving the tokens in the clock segment p to their target nodes (except when $p = |E(K)|$).

Using the vertex nodes corresponding to the subgraph of G isomorphic to K , we can accomplish the transformation by $|V(K)|$ clearing multi-slides of tokens on vertex nodes, $2Q$ multi-slides for each superedge, $(|E(K)| - 1)(Q + R)$ multi-slides for the clock gadget tokens, L^* multi-slides between source and target gadgets of supernodes, and finally $|V(K)|$ multi-slides to fill the vertex nodes. The total number of multi-slides will be $b = (|E(K)| - 1)(Q + R) + L^* + 2|V(K)| + 2Q|E(K)|$. $\square$

Before we prove the backward direction, we first show that for a yes-instance of LABELED UNDIRECTED TOKEN MOVING, the clock gadget behaves as required: the clock can be in only one position at a time (Lemma 33), “time” cannot go backwards (Lemma 34), and at any point in the transformation, we can have at most one superedge with an edge path free of tokens (Lemma 35).

Lemma 33. *If (H, S, T, b) is a yes-instance of LABELED UNDIRECTED TOKEN MOVING, then we cannot have a transformation during which Q_p and $Q_{p'}$ for integers $p, p' \in [|E(K)|]$ are such that the top ends of both are token-free.*

Proof of Lemma 33. If there exist K_p and $K_{p'}$ such that the top ends of both are free of tokens, then this implies that the tokens of at least one clock segment must multi-slide twice. Hence, the number of multi-slides required in the clock gadget will be at least $|E(K)|(Q + R) > |E(K)|R - 1 = b$. $\square$

Lemma 34. *If (H, S, T, b) is a yes-instance of LABELED UNDIRECTED TOKEN MOVING, then after the clock reaches position p for some $p \in [|E(K)|]$ in a transformation it can never go back to position $p - 1$ (or any earlier position).*

Proof of Lemma 34. Assume that the clock reaches position p and goes back to position $p - 1$. This implies that the clock would have to alternate from position p , to $p - 1$, and back to p . Hence, the total number of multi-slides to transform the tokens on the clock to their target positions would be $(|E(K)| + 1)(Q + R) > |E(K)|R - 1 = b$, a contradiction. $\square$

Lemma 35. *If (G, S, T, b) is a yes-instance of LABELED UNDIRECTED TOKEN MOVING, then at any point in the transformation we can have at most one superedge with an edge path whose nodes are free of tokens. Moreover, whenever a superedge clock numbered p has a token-free edge path, then the clock must be at position p .*

Proof of Lemma 35. By construction, each edge path in a superedge contains $Q = 2L^* + |V(K)| + 1$ tokens. Since the total number of nodes in source gadgets, target gadgets, and storage gadgets is $2L^* + |V(K)|$, at least one token of the token-free edge path must have moved to the clock gadget. Since the top of K_p is the only vertex in the clock gadget connected to the edge path, the top of K_p must have been token-free at the time of the multi-slide. $\square$

Lemma 36. *If (H, S, T, b) is a yes-instance of LABELED UNDIRECTED TOKEN MOVING, then $((G, \mathcal{C}_G), (K, \mathcal{C}_K))$ is a yes-instance of MULTICOLORED SUBGRAPH ISOMORPHISM.*

Proof of Lemma 36. In counting the total number of multi-slides in any transformation sequence, we need to allot at least $(|E(K)| - 1)(Q + R)$ multi-slides for transforming the clock gadget and at least L^* multi-slides of tokens from source gadgets to target gadgets. Moreover, since each vertex node blocks the multi-slides from source gadgets to target gadgets, at least $2c$ multi-slides are required to clear and then fill c node vertices. Of the b multi-slides in total, at most $2Q|E(K)|$ multi-slides remain. If we can show that all of these multi-slides are used to clear and fill an edge path in each of the superedges, then we can conclude that there must be a subgraph of G isomorphic to K . Hence, to complete the proof, it suffices to show that each superedge is used for a multi-slide, namely, that for all C_i and C_j , at least one D_i-D_j or D_j-D_i token passes through superedge $D_{i,j}$.

Suppose to the contrary that there exists a superedge $D_{i,j}$ that is not used for any multi-slide, and consider the first such superedge with the smallest clock number, say for D_i at level x and D_j at level $x + 1$. To determine the possible sequence of superedges in the paths for the multi-slides of the D_i-D_j tokens from D_i to D_j and the D_j-D_i tokens from D_j to D_i , we observe that Lemmas 33, 34, and 35 imply the following: the clock starts at position 1 and advances one position at a time; at any point in time, we can have at most one superedge with an edge-path free of tokens; and whenever superedge clock numbered p has a free edge-path, then the clock must be at position p . Due to the clock numbering of the superedges, we can view each superedge as *opening* and then *closing*; before opening and after closing, a superedge cannot be used for the movement of tokens. Furthermore, since the superedges open and close in order by level, we can refer to the *level x opening period* as the sequence of clock multi-slides during which each superedge of level x is opened and closed.

Since tokens can move only between supernodes at levels x and $x + 1$ during the level x opening period, a D_i - D_j token must move to level $x + 1$ by the end of the level x opening period, and then use at least one level $x + 2$ supernode (and possibly an alternating sequence of level $x + 1$ and $x + 2$ supernodes) during the level $x + 1$ opening period to reach D_j . As for any D_j - D_i token, it must move to its target position, i.e., D_i , using an alternating sequence of level x and level $x + 1$ supernodes. This also needs to happen by the end of the level x opening period.

We first show that D_i must have exactly two super-out-neighbors at level $x + 1$. Since no edge path in a superedge at level $x - 1$ or earlier can be opened in the future, the first edge in the path used by the D_i - D_j tokens is from D_i to a super-out-neighbor $D_b \neq D_j$ at level $x + 1$. As a consequence, D_i must have at least two super-out-neighbors at level $x + 1$. The supernode D_i cannot be the supernode at level 0, since there does not exist an alternating path of superedges at level 0 to be used to move the D_j - D_i tokens; this implies that D_i cannot have three super-out-neighbors.

During the level x opening period, the D_j - D_i tokens must move through a sequence of level x superedges, alternating between level x and $x + 1$ supernodes, until reaching D_i from super-out-neighbor $D_b \neq D_j$ at level $x + 1$. In order for the tokens to reach D_b from D_j , they must pass through a super-in-neighbor $D_a \neq D_i$ of D_b at level x (possibly after a long sequence of superedges). This implies that the superedge $D_{a,b}$ opens before the superedge $D_{i,b}$, allowing the tokens to move from D_a to D_b to D_i .

We now consider two cases, depending on whether D_b has two or three super-in-neighbors at level x .

Case 1: D_b has two super-in-neighbors. First, we show that it is impossible for both the D_i - D_j tokens to reach D_j and the D_j - D_i tokens to reach D_i during the level x opening period. Assume D_j has two super-in-neighbors at level x . Let D_c be the other neighbor of D_j at level x . Then, for the D_i - D_j tokens to move from D_i to D_j during the level x opening period, the superedge $D_{i,b}$ must open before the superedge $D_{c,j}$. For the D_j - D_i tokens to move from D_j to D_i during the level x opening period, the superedge $D_{c,j}$ must open before the superedge $D_{i,b}$, a contradiction. Since D_i has exactly two super-out-neighbors at level $x + 1$, the same argument applies when D_j has three super-in-neighbors at level x . We can then conclude that the D_i - D_j tokens arrive at a level $x + 1$ supernode, in particular the D_b supernode (since superedge $D_{a,b}$ opens before the superedge $D_{i,b}$), at the end of the level x opening period, and then multi-slide from D_b to D_j during the level $x + 1$ opening period.

Since the D_j - D_i tokens need to be able to move out of D_b and into D_j during the level $x + 1$ opening period, and since both D_j and D_b have two super-in-neighbors at level x ,

then both have one super-out-neighbor at level $x+2$. We denote these super-out-neighbors by $D_{j'}$ and $D_{b'}$, respectively. To multi-slide the D_i - D_j tokens to D_j at the opening of $D_{j,j'}$, all r_x tokens must be stored in $D_{j'}$. However, $D_{j'}$ has at most $10r_{x+1} + 1$ nodes in its source, target, and storage gadgets (one node for storage, at most $6r_{x+1}$ source and target nodes of tokens coming from or going to at most three super-in-neighbors, and at most $4r_{x+1}$ source and target nodes of tokens coming from or going to at most two super-out-neighbors), which is insufficient as $r_x = L \cdot r_{x+1} > 10r_{x+1} + 1$. Thus, it is not possible to multi-slide the D_i - D_j tokens in this case, as needed.

Case 2: D_b has three super-in-neighbors. As in the case with two super-in-neighbors, we first show that it is impossible for both the D_i - D_j tokens to reach D_j and the D_j - D_i tokens to reach D_i during the level x opening period. Recall that when clock numbering the superedges within a level, we are free to number them in any order. Hence, we assume that all superedges corresponding to a supernode at level $x+1$ (with its super-in-neighbors) must happen consecutively (for each supernode). This implies that all superedges incident to D_j will open after or before the D_i - D_j tokens have left D_i , a contradiction.

When D_b has three super-in-neighbors, we know that D_b has zero super-out-neighbors at level $x+2$ (K is such that each vertex has degree three). For the D_j - D_i tokens to move from D_j to D_a (w.l.o.g.) to D_b and finally to D_i during the level x opening, the superedge $D_{i,b}$ must open after the superedge $D_{a,b}$. This implies that by the end of the level x opening period, all the D_i - D_j tokens will be at supernode D_b or will have moved to other supernodes at level $x+1$. However, since D_b has no neighbors at level $x+2$, none of these tokens can remain at D_b by the end of the level x opening period. Let us denote by D_1, D_2 , up to D_c , the at most c supernodes at level $x+1$ that will contain the D_i - D_j tokens by the end of the level x opening period. The number of D_i - D_j tokens is $r_x = L \cdot r_{x+1}$. We note that all of these tokens must move to $D_{j'}$ prior to the edge $D_{j,j'}$ opening. However, $D_{j'}$ has at most $10r_{x+1} + 1$ nodes in its source, target, and storage gadgets, which is insufficient as $r_x = L \cdot r_{x+1} > 10r_{x+1} + 1$. Thus, it is again not possible to move the D_i - D_j tokens in this case.

This completes the proof. □

Combining Lemmas 32 and 36 with the fact that $b = \mathcal{O}(|V(K)|^{\mathcal{O}(|V(K)|)})$ and $f = \mathcal{O}(|V(K)|)$, we obtain the following theorem.

Theorem 37. *Labeled Undirected Token Moving is $W[1]$ -hard when parameterized by $b + f$.*

Chapter 5

Solution Discovery for Problems in P

In this chapter, we focus on all of MATCHING-DISCOVERY (Section 5.1), VERTEX/EDGE CUT-DISCOVERY (Section 5.2), and SHORTEST PATH-DISCOVERY (Section 5.3). That is, solution discovery problems where the properties are defined with well-studied polynomial-time solvable problems.

We study these problems under classical and parameterized complexity assumptions. In the latter case, we focus on the parameters k (that is, the number of tokens) and b (that is, the budget). For each problem that we prove to be fixed-parameter tractable with respect to some parameter, we show whether it admits a polynomial kernel with respect to the same parameter or whether such a kernel is unlikely under the assumption that $\text{NP} \not\subseteq \text{coNP}/\text{P}$. The results presented in this chapter appear in the papers titled:

- *Solution Discovery via Reconfiguration for Problems in P*, in proceedings of *ICALP 2024*, and co-authored with Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, and Sebastian Siebertz, and in
- *Kernelization Complexity of Solution Discovery Problems*, in proceedings of *ISAAC 2024*, and co-authored with Mario Grobler, Amer E. Mouawad, Naomi Nishimura, Vijayaragunathan Ramamoorthi, and Sebastian Siebertz.

Under classical complexity, we show that all three problems are NP-hard. We note that VERTEX CUT-DISCOVERY was already shown to be NP-hard but our proof is simultaneously a parameterized hardness proof. Similarly, SHORTEST PATH-DISCOVERY was shown to be NP-hard, but our proof holds for more restrictive graph classes. In

contrast to the results obtained by Fellows et al. [83] for VERTEX COVER-DISCOVERY, INDEPENDENT SET-DISCOVERY, and DOMINATING SET-DISCOVERY, our parameterized complexity results suggest that the parameter k consistently enables tractability across all three problems, suggesting that solution discovery becomes manageable when we bound the number of resources for these problems. The parameter b presents a different story: while limiting the number of allowed moves makes SHORTEST PATH-DISCOVERY tractable, it leaves MATCHING-DISCOVERY and VERTEX/EDGE CUT-DISCOVERY intractable. This asymmetry stems from the fact that SHORTEST PATH-DISCOVERY benefits from some distance-based structure that can be constructed between the two vertices of the input. In this structure, most tokens (assuming b is much smaller than k) must already be at their approximately correct distances within the structure, which implies only local adjustments with respect to b are required. In contrast, MATCHING-DISCOVERY and VERTEX/EDGE CUT-DISCOVERY lack this distance-based structure and demand more global reorganization, making the budget parameter insufficient to capture the problem’s complexity.

The kernelization landscape provides another lens into the structures of these problems. Particularly, there exists a polynomial kernel for MATCHING-DISCOVERY but not for VERTEX/EDGE CUT-DISCOVERY or SHORTEST PATH-DISCOVERY (even with combined parameters). The polynomial kernel for MATCHING-DISCOVERY benefits from the local nature of matching constraints, which can be checked and reduced independently within bounded neighborhoods—an advantage not shared by the more global connectivity requirements of cuts and paths.

Finally, recall that, under solution discovery, we denote a transformation sequence by $\vec{T}$ and use T for the final configuration. Additionally, the transformation sequence is formed of token slides.

5.1 Matching-Discovery

In this section, we focus on MATCHING-DISCOVERY. We show in Section 5.1.1 that the problem is NP-hard via a reduction from the RAINBOW MATCHING problem (which we also define in that section). In Section 5.1.2, we construct token-specific layer graphs that partition the vertices based on their distance from the token and show that the problem is fixed-parameter tractable when parameterized by k by proving that we can limit the search space for the destination edge of each token to the layers that are a bounded distance away from the token and that these layers can also be of a bounded size. It is important for this algorithm that we show that for this problem, b can always be set to at most $2(k^2 + k)$. In Section 5.1.3, we take a different approach toward kernelization and show how using the

Sunflower Lemma, we can prove that the problem admits a polynomial kernel with respect to the same parameter. Finally, in Section 5.1.4, we show that MATCHING-DISCOVERY is $W[1]$ -hard when parameterized by b through a reduction from the MULTICOLORED CLIQUE problem (which we define in the same section).

5.1.1 NP-hardness of Matching-Discovery

In this section, we show that MATCHING-DISCOVERY is NP-hard even on bipartite graphs. Thus, we prove the following theorem.

Theorem 38. MATCHING-DISCOVERY is NP-hard on bipartite graphs.

For this result, we reduce from the RAINBOW MATCHING problem. Let $(G, \mathcal{C})$ be an edge-colored graph where each edge has one color. A matching $\mathcal{M} \subseteq E(G)$ is said to be a *rainbow matching* if all edges in $\mathcal{M}$ have pairwise distinct colors. In the RAINBOW MATCHING problem, we are given a graph $(G, \mathcal{C})$ and an integer κ . The goal is to decide whether there exists a rainbow matching of size κ in $(G, \mathcal{C})$. An edge coloring of $E(G)$ is said to be *proper* if every pair of adjacent edges have distinct colors. The RAINBOW MATCHING problem is NP-complete, even when restricted to properly edge-colored paths [137].

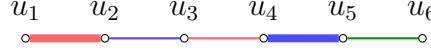
Let $(G, \mathcal{C}, \kappa)$ be an instance of RAINBOW MATCHING where G is a path, and let $r = |\mathcal{C}| - \kappa$. For clarity, we refer to *vertices* in the instance of RAINBOW MATCHING and *nodes* in our construction. To construct H as a bipartite graph, we first partition the vertices of G into two partite sets A and B . Note that for a path, this means the vertices in order alternate between being placed in A or B . The construction creates three types of gadgets.

For each edge $e = uv \in E(G)$, we will have an *edge path for uv* consisting of two nodes, a *u -node for e* and a *v -node for e* , connected by an edge.

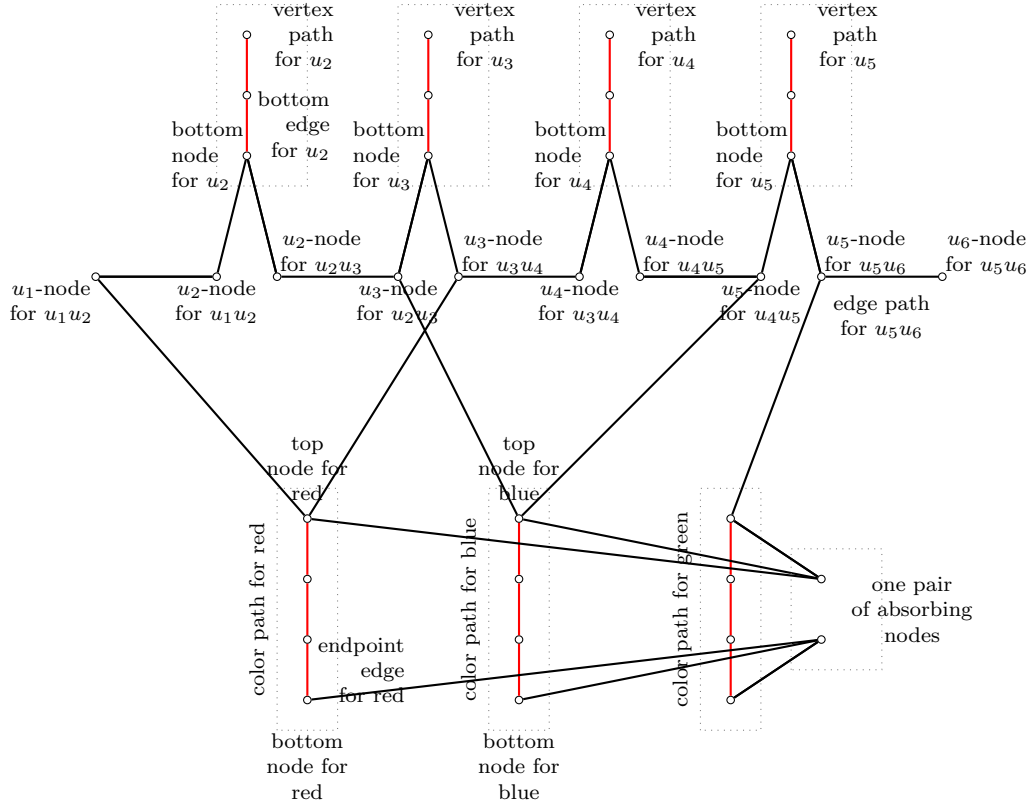
For each degree-2 vertex $v \in V(G)$, we will have a *vertex path for v* consisting of three nodes such that the *bottom node for v* (that is, one endpoint of the vertex path for v that belongs to the *bottom edge for v*) is adjacent to the *u -nodes* for both edges incident to u .

For each color $C \in \mathcal{C}$, we will have a *color path for C* consisting of four nodes (that is, three edges) such that the *top node for C* (that is, one endpoint of the color path for C) is adjacent to the *u -node* for each edge $e = uv$ where $u \in A$ and e has the color C or the *v -node* for each edge $e = uv$ where $v \in A$ and e has the color C . We refer to the other endpoint of the color path for C as the *bottom node for C* . We refer to each edge of the color path for C that has one of the top or bottom nodes for C as an *endpoint edge for C* . See Figure 5.1.

Initially, tokens are placed on all edges of all vertex and color paths, thus each creating an edge set that does not form a matching, which we will call a “broken” matching that must be repaired. We will say we “fix” the vertex or color path to mean we fix its broken matching by leaving tokens only on some edge subset of the path edges that forms a matching.



(a) Input properly edge-colored path $(G, \mathcal{C})$. $|\mathcal{C}| = 3$. A rainbow matching is marked with thick edges. The vertices u_1 , u_3 , and u_5 form the partite set A .



(b) The reduced graph H for $(G, \mathcal{C})$ with $r = 1$ and $\kappa = 2$. Initial placement of tokens (edges in S) is shown in red.

Figure 5.1: Reduction from RAINBOW MATCHING to MATCHING-DISCOVERY.

Note that each color path can be fixed by either losing the token on its middle edge

or both tokens on its endpoint edges. We couple this fact with a tight budget in order to encode the choice of which edges of $(G, \mathcal{C})$ are in the rainbow matching. For that we first add r pairs of *absorbing nodes* (each pair having a *top* and a *bottom* absorbing node) adjacent to the top and bottom nodes for every C in $\mathcal{C}$. Each such pair of nodes can “absorb” two tokens from the endpoint edges of the color paths in one token slide per token, and thus can fix r color paths using at most $2r$ token slides.

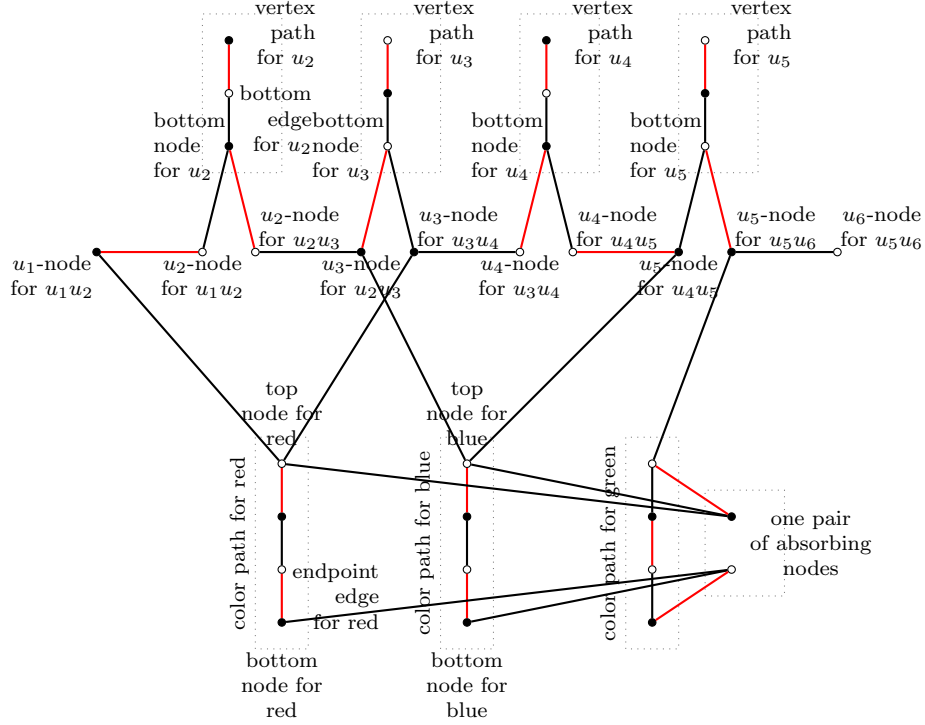


Figure 5.2: The reduced graph H . The red edges show the final matching configuration after all token movements. The filled and unfilled nodes show that the graph is bipartite. The vertices u_1 , u_3 , and u_5 form the partite set A .

Second, the reduction employs a tight budget constraint as we set $b = 3\kappa + 2r + n - 2$ which forces a specific movement pattern: each vertex path for a vertex $u \in V(D)$ requires exactly one token slide to fix (moving the token on the bottom edge for u to one of the edge paths representing edges incident to u in G) (see Figure 5.2), while each color path requires either:

- the *2-slides option*: that is, sliding each of the two tokens on its endpoint edges onto an edge incident to an unsaturated absorbing node, or

- the *3-slides option*: that is, moving the token on its middle edge to an edge path whose endpoints are not already saturated by a token from a vertex path. This requires three token slides.

The budget allows exactly κ color paths to use the 3-slides option and go to edge paths representing edges of their color, and each token sliding from a vertex path ensures that only one of two edge paths representing adjacent edges in G can receive a token as it saturates an endpoint of one of the edge paths. Thus, the edge paths having tokens at the end will form a rainbow matching in $(G, \mathcal{C})$ of size κ .

It is clear that the reduction runs in polynomial time with respect to $|V(G)|$. Next, we present in greater detail the forward and backward directions of the proof of Theorem 38. Before that, we start by showing that H is a bipartite graph.

Lemma 39. *The graph H is a bipartite graph.*

Proof of Lemma 39. Let (A, B) be a bipartition of the path G . We construct a bipartition (L, R) of $V(H)$ as follows.

- For each edge path for uv we have $u \in A$ and $v \in B$, and thus we place the u -node for uv in L and the v -node for uv in R .
- For each vertex path for $w \in V(G)$, we place the bottom node and the top node for w in a different partition than the one the w -nodes for edges incident to w are in, and we do the opposite for the middle node of the vertex path for w .
- For each color path for $c \in \mathcal{C}$, we place the top node for C in R and we alternate the partition assignment along the color path, placing the bottom node for C in L .
- We place all top absorbing nodes in L and all bottom absorbing nodes in R .

Since G is a bipartite path, and our construction preserves this bipartition structure, H is bipartite. $\square$

Lemma 40. *If $(G, \mathcal{C}, \kappa)$ is a yes-instance of RAINBOW MATCHING, then (H, S, b) is a yes-instance of MATCHING-DISCOVERY.*

Proof of Lemma 40. Let $R \subseteq E(G)$ be a rainbow matching of size κ in $(G, \mathcal{C})$. We show how to move tokens to create a valid matching in H within budget $b = 3\kappa + 2r + n - 2$.

For each degree-2 vertex v in G , take the token on the bottom edge of the vertex path for v and slide it to one of the edge paths representing an edge incident to v that is not in R . Since R is a matching, at least one such edge path exists. This costs one token slide per vertex path, totaling $n - 2$ token slides.

For each edge $e \in R$ with color C , take the token on the middle edge of the color path for C and slide it to the edge path for e (that is, the 3-slides option). This costs three token slides per color, totaling 3κ token slides.

There are exactly $r = |\mathcal{C}| - \kappa$ remaining color paths with three tokens. For each such color path, take the token on the top endpoint edge and slide it toward any unsaturated top absorbing node (that is, the 2-slides option), and take the token on the bottom endpoint edge and slide it to any unsaturated bottom absorbing node (that is, the 2-slides option). This costs two token slides per color path, totaling $2r$ token slides.

It is easy to verify that the resulting set of tokens constitutes a matching in H . The total cost to get to this matching is $(n - 2) + 3\kappa + 2r = b$. Therefore, (H, S, b) is a yes-instance of MATCHING-DISCOVERY. $\square$

Lemma 41. *If (H, S, b) is a yes-instance of MATCHING-DISCOVERY, then $(G, \mathcal{C}, \kappa)$ is a yes-instance of RAINBOW MATCHING.*

Proof of Lemma 41. Let T be a matching in H obtainable from S using at most $b = 3\kappa + 2r + n - 2$ token slides.

First, we show that any feasible solution uses exactly b token slides as follows. Each vertex path requires at least one token slide to fix (sliding a token from the bottom edge). Each color path requires at least two token slides to fix. Since there are $n - 2$ vertex paths and $|\mathcal{C}| = \kappa + r$ color paths, we need at least $(n - 2) + 2(\kappa + r)$ token slides. However, at most r color paths can be fixed using exactly two token slides (by moving both endpoint tokens to absorbing nodes), as there are only r pairs of absorbing nodes. The remaining κ color paths require at least three token slides each. Therefore, any feasible solution needs at least $(n - 2) + 2r + 3\kappa = b$ slides, which corresponds to the budget.

Now we analyze how the tokens move. For each vertex v with $\deg_G(v) = 2$, the token from the bottom edge of the vertex path for v must slide to an edge connecting the bottom node for v to a v -node for some edge incident to v . This uses exactly one slide per vertex path. Since each bottom node for v can only have one token on an incident edge, at most one edge path for edges incident to v can have a token on it.

Since we use exactly b slides, exactly r color paths use the 2-slides option (moving tokens on endpoint edges to edges incident to two absorbing nodes), and the remaining

κ color paths use three token slides each. For a color path to use three token slides, its middle token must slide to an edge path (as bottom nodes of vertex paths are all saturated and cannot slide with zero token slides remaining in the budget). The only way to reach an edge path in exactly three token slides from the middle edge of a color path for color C is: from middle edge of the color path for C to top edge for C and then to edge path for e where the color of e is C .

Let $R = \{e \in E(G) \mid \text{the edge path for } e \text{ receives a token from a color path}\}$. Since tokens from vertex paths ensure that at most one edge path representing an edge incident to a vertex can receive a token, R forms a matching in G . Moreover, each of the κ color paths using the 3-slides option sends its token to an edge path for an edge of its corresponding color, and since these κ colors are all distinct, the edges in R have distinct colors. Therefore, R is a rainbow matching of size κ in $(G, \mathcal{C})$. $\square$

5.1.2 Fixed-parameter tractability of Matching-Discovery for Parameter k

In this section, we show that MATCHING-DISCOVERY is fixed-parameter tractable for parameter k . We do so by proving first that MATCHING-DISCOVERY is fixed-parameter tractable when parameterized by $k+b$. Then, we show that for any instance of the problem one can bound b by a (quadratic) function of k which implies fixed-parameter tractability of the problem when parameterized by k alone. We start with some relevant definitions and lemmas.

In what follows, for a token t on an edge $e \in S$ and a vertex $v \in V(G)$, $b(e, v)$ denotes the minimum number of token slides that the token initially on $e \in S$ must slide in order to become incident to the vertex v .

Definition 42. For an integer $i \in \{1, \dots, b\}$ and an edge $e \in S$, we let Y_e^i denote the set $\{w \in V(G) \mid b(e, w) = i - 1\}$ and let Z_e^i denote the set $\{w \in V(G) \mid b(e, w) = i\}$. Let G_e^i denote the graph $G[Y_e^i \cup Z_e^i] - E(G[Z_e^i])$.

Informally, the sets Y_e^i and Z_e^i partition vertices by their distance from edge e : those reachable in $i - 1$ token slides versus i token slides. The *layer graph* or graph G_e^i preserves all the edges between the vertices of layer Y_e^i and each edge between a vertex from Y_e^i and a vertex from Z_e^i .

The following lemma establishes a key property for bounding our search space for a solution: if a token ends up in the final configuration on an edge in a layer graph with

one endpoint having many incident edges in the layer graph (at least $2k - 1$), then we can always find an alternative edge in the layer graph incident to that high-degree vertex that yields an equally valid solution with the same number of token slides. The intuition is that if we know that some token will go to an edge in a specific layer graph, then for each vertex in that layer graph, we need to consider at most $2k - 1$ incident edges as potential destinations for that token in the solution.

Lemma 43. *Let (G, S, b) be a yes-instance of MATCHING-DISCOVERY and $(T, \vec{T})$ be a feasible solution that uses the minimum number of token slides. Let $b' \leq b$ be the number of token slides in $\vec{T}$ used by some token t on an edge $e \in S$ to reach an edge uv in T . Note that both u and v as well as the edge uv are contained in $G_e^{b'}$. If u or v has at least $2k - 1$ neighbors in $G_e^{b'}$, then there exists at least one neighbor, say w , such that sliding t to uw or wv (and leaving the final positions of all other tokens unchanged) produces another feasible solution that requires the same number of token slides.*

Proof of Lemma 43. Assume, without loss of generality, that $u \in Y_e^{b'}$. We know that after applying the token slides in $\vec{T}$, t becomes incident to u and v in $G_e^{b'}$. We also know that a single token can saturate at most two vertices. Thus, $k - 1$ tokens can saturate at most $2k - 2$ vertices. This implies that if all tokens initially on the edges in $S \setminus \{e\}$ end up on edges in T that are incident to some vertices in $N_G(u) \cap (Y_e^{b'} \cup Z_e^{b'})$, they can saturate at most $2k - 2$ of those vertices.

If u has more than $2k - 2$ (at least $2k - 1$) neighbors in $G_e^{b'}$, there must exist one unsaturated vertex w that is a neighbor of u and that is not an endpoint of an edge in T . Thus, we can form a new transformation sequence that slides t to uw and slides all tokens initially on the edges of $S \setminus \{e\}$ to the edges for which they are originally destined in $(T, \vec{T})$. This produces a valid matching in G as w is not an endpoint of an edge in T . Notice that since $u \in Y_e^{b'}$, the number of token slides needed to slide t to uv is equal to the number of token slides needed to slide t to uw . Thus, the transformation sequence requires the same number of token slides in $\vec{T}$.

A similar argument holds if v has more than $2k - 2$ neighbors in $G_e^{b'}$, as all of these neighbors must belong to $Y_e^{b'}$ (recall that we delete the edges between the vertices in $Z_e^{b'}$ when building $G_e^{b'}$), which completes the proof. $\square$

The next lemma provides another crucial bound on our search space. It shows that when a layer graph contains a large matching (of size at least $2k$), we can restrict our attention to just $2k$ edges from this matching as potential destinations for any token in the solution.

Lemma 44. *Let (G, S, b) be a yes-instance of MATCHING-DISCOVERY and $(T, \vec{T})$ be a feasible solution that uses the minimum number of token slides. Let $b' \leq b$ be the number of slides in $\vec{T}$ used by some token t on an edge in $e \in S$ to reach an edge e' in T . If there is a matching M of size at least $2k$ in $G_e^{b'}$, then there exists an edge $e'' \neq e' \in M$ such that sliding t to e'' while sliding all other tokens initially on edges in $S \setminus \{e\}$ to the edges for which they are originally destined in $(T, \vec{T})$ produces another feasible solution.*

Proof of Lemma 44. Given that in the original solution, t slides b' steps, we know that t ends up on one of the edges in $G_e^{b'}$. In fact, the edge e' has both endpoints in $Y_e^{b'}$ or one endpoint in $Y_e^{b'}$ and the other in $Z_e^{b'}$. We also know that a single token can saturate at most two vertices, thus $k - 1$ tokens can saturate at most $2k - 2$ vertices. This implies that if all tokens initially on edges in $S \setminus \{e\}$ end up on edges in T incident to vertices in $G_e^{b'}$, they can saturate at most $2k - 2$ vertices in that subgraph. Each of those $2k - 2$ vertices can appear as an endpoint of one matching edge in M . Thus, since M has size at least $2k$, there must exist at least one edge $e'' \neq e'$ in M that is not incident to any of the vertices that are saturated after all tokens on edges in $S \setminus \{e\}$ move to the edge for which they are originally destined in $(T, \vec{T})$. Thus, we slide t to e'' and slide all tokens initially on the edges of $S \setminus \{e\}$ to the edges for which they are originally destined in $(T, \vec{T})$. This produces a valid matching in G as e'' is not incident to any of the other vertices saturated by the tokens on the edges in $S \setminus \{e\}$. Since it also requires the same number of token slides to move t to e'' or to e' , the new transformation sequence requires the same number of token slides in $\vec{T}$. $\square$

We now present the main algorithm that exploits the structural properties established in the previous lemmas.

Theorem 45. MATCHING-DISCOVERY parameterized by $k + b$ is fixed-parameter tractable.

Proof of Theorem 45. Let (G, S, b) be an instance of MATCHING-DISCOVERY. The algorithm proceeds as follows. We first guess a set $WS \subseteq S$ of edges whose tokens will slide from their original positions. If such a guess leaves any vertices incident to more than one edge having a token, we can ignore it and proceed to the next guess. Note that this guessing procedure requires $\mathcal{O}(2^k)$ time in the worst case. Next, for each $b' \in \{0, 1, \dots, b\}$ and for a fixed WS , we guess a partition of b' over the tokens on the edges of WS . In other words, we try all possible ways of distributing the budget b' over the tokens in WS .

Now, for a fixed subset WS , a fixed b' , and a fixed distribution D of b' over the tokens on the edges of WS , we let b'_i be the budget allocated to a token t_i (initially on an edge

$e_i \in WS$) under D . The algorithm computes the sets $Y_{e_i}^{b'_i}$ and $Z_{e_i}^{b'_i}$ for each token t_i . The next step is to produce a set of candidate edges for each t_i , which we denote by T_i . A candidate edge is one possible edge that t_i might be destined for in our solution. To produce T_i , we run a maximum matching algorithm in the graph $G_{e_i}^{b'_i} - (S \setminus WS)$. There are two cases to consider:

- If, for t_i , the algorithm finds a maximum matching M in $G_{e_i}^{b'_i} - (S \setminus WS)$ such that $|M| \geq 2k$, then we set T_i to any $2k$ edges of M .
- If $|M| < 2k$, then $|V(M)| \leq 4k - 2$ and $G_{e_i}^{b'_i} - (S \setminus WS)$ has a vertex cover $V(M)$ of size at most $4k - 2$. This is true because if there is an edge in $G_{e_i}^{b'_i} - (S \setminus WS)$ that is not incident to any of the $4k - 2$ vertices of M , then M is not a maximum matching. When $|E(G_{e_i}^{b'_i} - (S \setminus WS))| \leq (4k - 2)(2k - 1)$, we simply set $T_i = E(G_{e_i}^{b'_i} - (S \setminus WS))$. Otherwise, we let T_i include M as well as, for each vertex of the vertex cover, at most $2k - 1$ incident and distinct edges of $G_{e_i}^{b'_i} - (S \setminus WS)$.

Once the candidate sets have been constructed, the algorithm exhaustively checks (by guessing for each token t_i on an edge in WS the edge in T_i it is destined for in the solution). whether it can reach a valid matching in the graph G , and if so, terminates with a YES. The algorithm terminates with a no if no valid matching is reached during this search.

The correctness of the algorithm follows from Lemmas 43 and 44 and the fact that we exhaustively try candidate edges for each token. In particular, if a solution exists, the algorithm must find a subset $WS \subseteq S$ of tokens that move, and the values b'_i for each t_i on an edge $e_i \in WS$ that represent the number of slides performed by each token t_i . Given Lemma 44, we know that for a certain e_i , if the size of a maximum matching M is at least $2k$, then we can find an edge $e'' \in M$ to slide t_i to no matter which edges the other tokens that slide in the solution are destined for. If for a certain e_i the maximum matching algorithm returns a matching M of size less than $2k$, then, by the definition of a maximum matching, we know that each edge in $G_{e_i}^{b'_i} - (S \setminus WS)$ is incident to at least one of the endpoints of the edges in M . There are at most $4k - 2$ endpoints and we know that the token t_i will slide in the solution to an edge incident to one of them. By Lemma 43, if such an endpoint (which we denote by u) has at least $2k - 1$ neighbors in $G_{e_i}^{b'_i} - (S \setminus WS)$, then we know that we can find an edge $e = uw$ to slide t_i to no matter which edges the other tokens that slide in the solution are destined for.

It is not hard to see that this algorithm runs in fixed-parameter tractable time. It loops $b + 1$ times and in each iteration, it chooses one of 2^k subsets of the k tokens to move. It

also distributes the budget $b' \leq b$ on this subset, thus far having $\binom{b+k-1}{k-1}$ choices in total per iteration. For each such choice, the algorithm runs, up to k times, the maximum matching procedure in $\mathcal{O}(\sqrt{n} \cdot m)$ time (using the algorithm of Vazirani [176]). Additionally, for each such choice, it encounters up to $\mathcal{O}(k)^2$ candidate edges per moving token. Checking whether the constructed solution is valid can be performed in time $\mathcal{O}(k^2)$. The whole algorithm thus runs in fixed-parameter tractable time with respect to k and b (namely, $\mathcal{O}\left((2k^2(b+k))^k \cdot n^{\mathcal{O}(1)}\right)$). $\square$

We now show that the same algorithm is also a fixed-parameter tractable algorithm for the parameter k alone by proving that k is an upper bound on the parameter b . For that, we first recall Theorem 8 from the preliminaries. The theorem states that a matching M in a graph G is a maximum matching if and only if G has no M -augmenting path, that is, a path that starts and ends on free (unsaturated) vertices and alternates between edges in M and edges not in M .

Lemma 46. *In any yes-instance (G, S, b) of MATCHING-DISCOVERY, b can be bounded from above by $2(k^2 + k)$.*

Proof of Theorem 46. We prove this lemma by induction on the number of tokens in an instance (G, S, b) of MATCHING DISCOVERY with $n = |V(G)|$ and $m = |E(G)|$. Let $\mathcal{P}(i)$ be the proposition that if (G, S, b) is a yes-instance of MATCHING-DISCOVERY with $|S| \leq i$, $|V(G)| = n$, and $|E(G)| = m$, then there exists a solution to (G, S, b) using at most $2(i^2 + i)$ token slides. The statement holds trivially for $i = 1$. Hence, assume $\mathcal{P}(i)$ is true for an arbitrary integer $i > 1$. We show that this implies that $\mathcal{P}(i + 1)$ is also true.

Given a yes-instance (G, S, b) with $|V(G)| = n$, $|E(G)| = m$ and $|S| = i + 1$, we form an instance $(G, S \setminus \{e\}, b)$ by deleting an arbitrary edge e from S . By the induction hypothesis, $(G, S \setminus \{e\}, b)$ is clearly a yes-instance and has a solution with at most $2(i^2 + i)$ token slides.

Given that (G, S, b) is also a yes-instance, we know that the matching in $(G, S \setminus \{e\}, b)$ is not maximum and, by Theorem 8, an augmenting path P of at most $\ell = 2i + 1$ edges exists in G after solving $(G, S \setminus \{e\}, b)$. P is formed by at most i tokens in $(G, S \setminus \{e\}, b)$. Let $\vec{T}^-$ be the transformation sequence of a feasible solution to $(G, S \setminus \{e\}, b)$. The edge e of (G, S, b) is either an edge of P or not. After applying the token slides in $\vec{T}^-$ on (G, S, b) , if e is an edge of P , at most $i + 1$ token slides are needed in addition to the $2(i^2 + i)$ token slides in $\vec{T}^-$ to get to a valid matching in (G, S, b) ; thus $b \leq 2(i^2 + i) + i + 1$. This is true because if e contains only one token, then we need to slide each of the other tokens on P once and if it contains two, then we need to slide one token on e to an incident edge and then slide each other token on P once. If e is not an edge of P , the token can be at most

$2i + 1$ edges away from the path P (since otherwise, we can find an augmenting path on the path that connects the token to P) and in that case, at most $3i + 3$ other token slides ($2i + 2$ to get the token to an edge of P) are needed to get to a valid matching in (G, S, b) ; thus $b \leq 2(i^2 + i) + 3i + 2$. In both cases, we form a solution to (G, S, b) that uses at most $2((i + 1)^2 + (i + 1))$ token slides. Since the choice of n and m was arbitrary, the statement holds for all yes-instances for any n and m . $\square$

Combining Theorem 45 with Lemma 46 we get the following result.

Corollary 47. *MATCHING-DISCOVERY is fixed-parameter tractable when parameterized by k .*

5.1.3 Polynomial Kernel for Matching-Discovery with Parameter k

In this section, we show that MATCHING-DISCOVERY admits a polynomial kernel when the parameter is k .

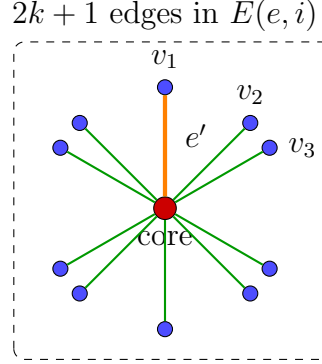
First, recall from Section 3.4 of the preliminaries that, for an instance with a given budget b , we call an edge $e \in E(G)$ *irrelevant with respect to an edge $e' \in S$* , if given a solution $(T, \vec{T})$ where the token on e' slides and stays on the edge $e \in T$, we can construct a solution that uses the same number of token slides and where e is not in the final configuration.

Hereafter, for edges $e, e' \in E(G)$, we denote by $ts(e, e')$ the minimum number of token slides needed to get a token on edge e to slide to the edge e' . For an edge $e \in E(G)$, we denote by $E(e, i) = \{e' \in E(G) \mid ts(e, e') = i\}$ (that is, $E(G_e^i)$ from the previous section). Before presenting the kernelization, we note that the following holds.

Claim 48. *In any solution $(T, \vec{T})$ to (G, S, b) , if a token on an edge $e \in S$ is destined for an edge $e' \in T$ such that $ts(e, e') > 3k$, it can instead be destined for an edge $e'' \in E(G)$ such that $ts(e, e'') \leq 3k$, while each other token on an edge in $S \setminus \{e\}$ remains destined for the same edge it was destined for in $(T, \vec{T})$, while maintaining feasibility.*

Proof of Claim 48. The edges in $T \setminus \{e'\}$ can appear in at most $k - 1$ of the $3k$ sets of edges $E(e, i)$ for $i \in [3k]$ and every edge that appears in a set $E(e, i)$ can only be incident to edges in $E(e, i - 1)$ and $E(e, i + 1)$. This implies that the token initially on e cannot be destined for any edge in as many as $3k - 3$ of the $3k$ sets $E(e, i)$ for $i \in [3k]$ (since these could result in incident tokens in the final configuration). Thus three of the $3k$ sets $E(e, i)$

for $i \in [3k]$ are token-free and are such that their adjacent edges are token-free, and there exists an edge e'' which the token on e can be destined for while each other token on an edge in $S \setminus \{e\}$ remains destined for the same edge it was destined for in $(T, \vec{T})$. Note that the new transformation sequence uses fewer token slides. $\square$



Each edge $e_i = \{\text{core}, v_i\}$ is a “set” in the sunflower
The core is the common vertex core

Figure 5.3: Sunflower structure in Theorem 49: The “sets” are edges (shown in green) that all share a common vertex that is the sunflower core. Each edge has one unique vertex v_i (the petals). The proof shows that if we have $2k + 1$ such edges, at least one can be used to replace e' in the matching.

Theorem 49. MATCHING-DISCOVERY parameterized by k admits a polynomial kernel with reduced instances of size $\mathcal{O}(k^5)$.

Proof of Theorem 49. Let (G, S, b) be an instance of MATCHING-DISCOVERY. Without loss of generality, we assume that the graph G is connected. For each edge $e \in S$, and integer $i \in [3k]$, we compute $E(e, i)$ in polynomial time. We maintain the invariant that we remove from $E(e, i)$, for each $e \in S$ and $i \in [3k]$, irrelevant edges with respect to e .

While $|E(e, i)| > 8k^2$ for an edge $e \in S$ and an integer $i \in [3k]$, we remove an irrelevant edge with respect to e as follows. From the Sunflower Lemma (Lemma 12), if $|E(e, i)| > 8k^2$, then it has a sunflower with $2k+1$ sets (edges) that can be computed in time polynomial in k . See Figure 5.3. We arbitrarily choose one edge e' corresponding to one set of the sunflower and remove it from $E(e, i)$.

To see why e' is irrelevant with respect to e , assume that the token on e is destined for e' . The $2k - 2$ vertices of the edges in $T \setminus \{e'\}$ can be incident to at most $2k - 2$ of

the edges corresponding to the sets of the sunflower. Additionally, none of these $2k - 2$ vertices is in the core of the sunflower. This leaves at least one set e'' that can replace e' in the matching T in G . Since all edges in $E(e, i)$ are at the same token sliding distance i from e , replacing e' by e'' will not increase the number of token slides needed to achieve $T \setminus \{e'\} \cup \{e''\}$.

We form the kernel (G', S, b') of the original instance (G, S, b) as follows. Given the above and Claim 48, we set $E(G') = \bigcup_{e \in S, i \in [3k]} E(e, i) \cup S$ and for each edge $e' \in E(e, i)$, for $e \in S$ and $i \in [3k]$, we add to $E(G')$ at most i edges that are on the shortest path from e to e' (if such edges are not already in $E(G')$). G' is the subgraph of G induced by the edges in $E(G')$. We set b' to the minimum of b and $k \cdot |V(G')|$.

By the end of this process, $|E(G')| \leq k + 24k^4 + 24k^4 \cdot (3k - 1)$, as for each $e \in S$ and $i \in [3k]$, $|E(e, i)| \leq 8k^2$, and for each edge of these $3k^2$ sets of edges, we added to $E(G')$ at most $3k - 1$ edges that are on a shortest path from that edge to the edge e . Additionally, constructing the new instance requires only time polynomial in n . Finally, (G', S, b') is a kernel since only edges that are irrelevant with respect to every token in S might not be in $E(G')$ and all edges needed to move tokens from edges in S toward a matching using at most b token slides are present in $E(G')$. Additionally, by Claim 10, the budget we need in G' is at most b' . $\square$

5.1.4 W[1]-hardness of Matching-Discovery for Parameter b

In this section, we show that MATCHING-DISCOVERY is W[1]-hard when parameterized by b even on 3-degenerate graphs. Thus, we prove the following theorem.

Theorem 50. *MATCHING-DISCOVERY is W[1]-hard on 3-degenerate graphs when parameterized by b .*

For this result, we reduce from the MULTICOLORED CLIQUE problem, which is W[1]-hard when parameterized by the solution size κ [58]. In the MULTICOLORED CLIQUE problem, we are given a graph G and an integer κ , where $V(G)$ is partitioned into κ color classes $\{V_1, \dots, V_\kappa\}$. The goal is to decide whether there exists a clique that contains one vertex from each color class. Without loss of generality, we will assume, throughout our reductions, that for each $i \in [\kappa]$, $|V_i| = n$ (we can add isolated vertices as needed). In an instance (G, κ) of the MULTICOLORED CLIQUE problem, the edge set $E(G)$ can be partitioned into $\binom{\kappa}{2}$ sets $E_{i,j} = \{uv \mid u \in V_i, v \in V_j, uv \in E(G)\}$ for $1 \leq i < j \leq \kappa$.

Let (G, κ) be an instance of MULTICOLORED CLIQUE. At a high level, the instance (H, S, b) of MATCHING-DISCOVERY encodes both vertex selection and clique verification

through carefully controlled token movements with an extremely tight budget. For clarity, we refer to *vertices* in the instance of MULTICOLORED CLIQUE and *nodes* in our construction.

The construction creates a *vertex-block* for each color class and an *edge-block* for each unordered pair of color classes.

Vertex-block. For a color class V_i , a *vertex-block for i* consists of a length-2 *vertex decider path* having one endpoint designated as the *vertex decider node*, and one vertex-gadget for each vertex in V_i .

The vertex decider path initially contains two tokens (one on each of its edges) and must evacuate one of them. Inside the vertex-block, this evacuation can happen in two ways (detailed below), with budget constraints making the efficient option necessary.

A *vertex-gadget for v* consists of a *vertex-gadget path* formed by concatenating $\kappa - 1$ *switch-gadgets*, one for each $j \neq i$ where V_j is a color class and $v \in V_i$.

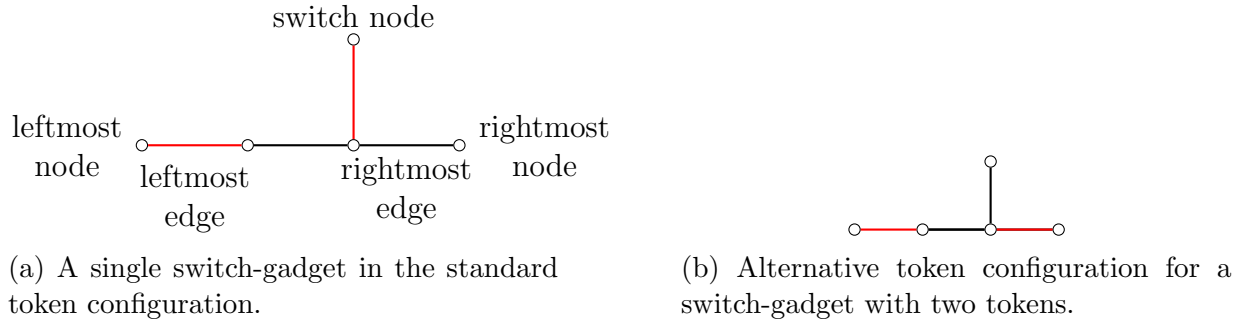


Figure 5.4: Switch-gadget structure and token configurations.

A *switch-gadget for (v, j)* is a subgraph with a path segment of three edges connecting four nodes and a pendant node, called the *switch node (v, j)* , attached to the third node of this path segment. We refer to the endpoints of a path segment as the *leftmost node for (v, j)* and *rightmost node for (v, j)* . We let the edge of the path segment incident to the rightmost node for (v, j) be the *rightmost edge for (v, j)* and that incident to the leftmost node for (v, j) be the *leftmost edge for (v, j)* . See Figure 5.4a. Each switch-gadget starts with a specific token placement called the *standard token configuration* that places a token on the edge incident to the switch node and another token on its leftmost edge. If required to have two tokens, a switch-gadget can undergo only a controlled state change to the *alternative token configuration* (see Figure 5.4b).

The vertex-gadget for v is constructed by connecting the $\kappa - 1$ switch-gadgets end-to-end. Specifically, for consecutive switch-gadgets, we add an edge between the rightmost node of one switch-gadget and the leftmost node of the next switch-gadget. After connecting all $\kappa - 1$ switch-gadgets, we add a final edge incident to the rightmost node of the last switch-gadget, with its other endpoint being a fresh degree-1 node called the *vertex select node for v* . The leftmost node of the first switch-gadget in the vertex-gadget for v is called the *vertex node for v* . The vertex decider node of the vertex-block for i is incident to the *vertex node* for each $v \in V_i$. See Figure 5.5.

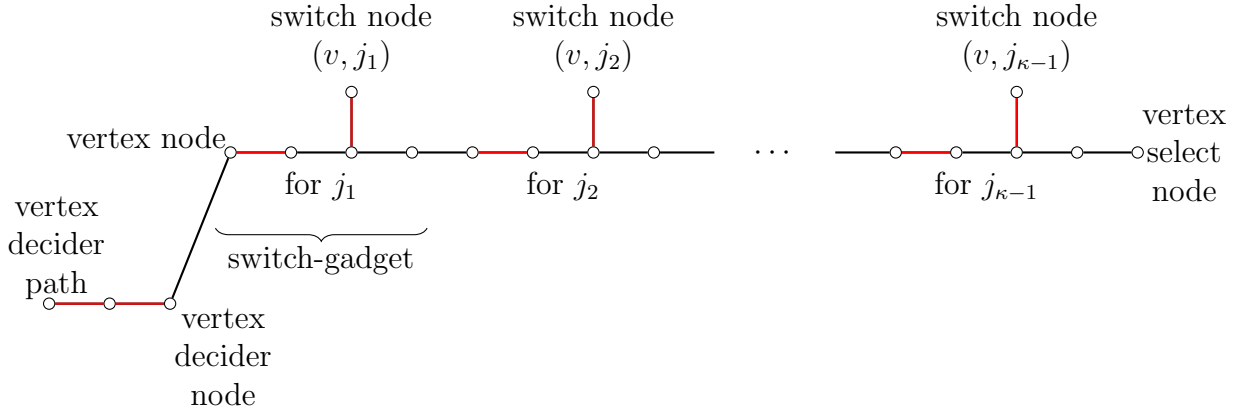


Figure 5.5: Vertex-gadget for v formed by concatenating $\kappa - 1$ switch-gadgets and connected to the vertex decider path. Each switch-gadget corresponds to an index $j \neq i \in [\kappa]$. The vertex node is the leftmost node of the gadget and the vertex select node is the rightmost node. Red edges highlight the initial placement of tokens.

The vertex decider path can evacuate its token in two ways within the vertex-block:

1. **Cascade evacuation I (expensive):** A vertex decider path token slides to the edge between the vertex decider node and the vertex node of a vertex-gadget, then triggers a cascade where each token in the gadget not on an edge incident to a switch node shifts three edges rightward along the path for a total of $3(\kappa - 1)$ token slides.

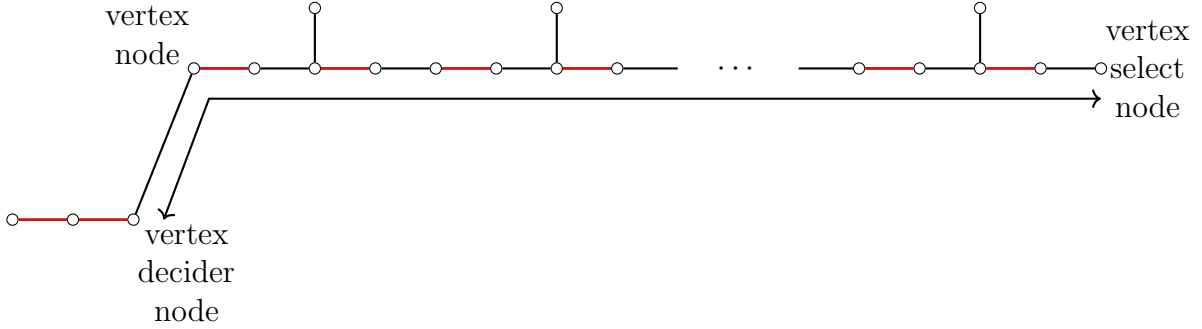


Figure 5.6: When all switch-gadgets are in the alternative configuration, an augmenting path (shown using the arrow) exists between the vertex decider node and the vertex select node.

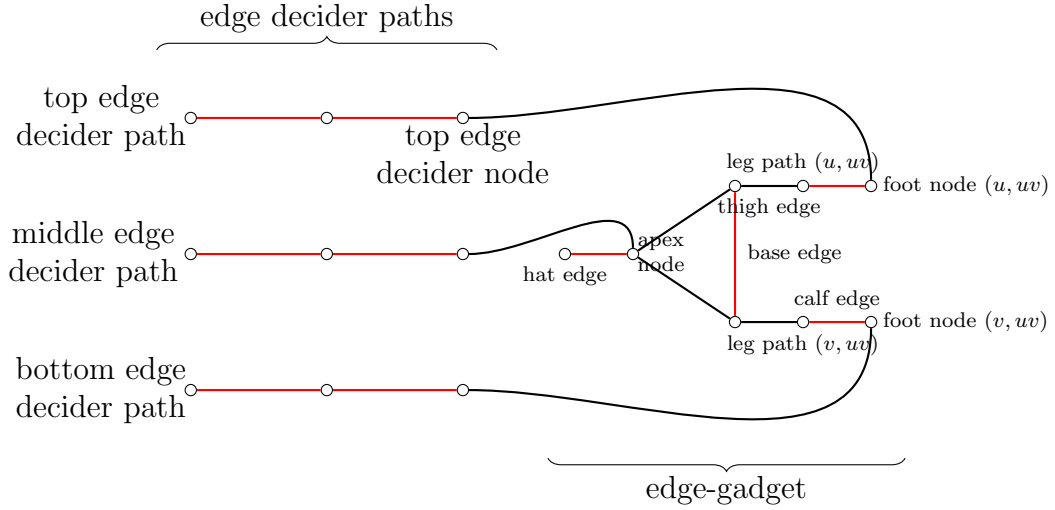
2. **Cascade evacuation II (efficient):** If each switch-gadget of some vertex-gadget in the vertex-block is in the alternative token configuration, we have an *augmenting path* between the vertex decider node and the vertex select node of the vertex-gadget (see Figure 5.6). The token can slide once to the edge incident to the vertex node of this vertex-gadget, and trigger a cascade where each token in the gadget not on an edge incident to a switch node shifts one position right. This uses $\kappa - 1$ fewer slides than Cascade evacuation I.

Budget constraints will make the second option necessary.

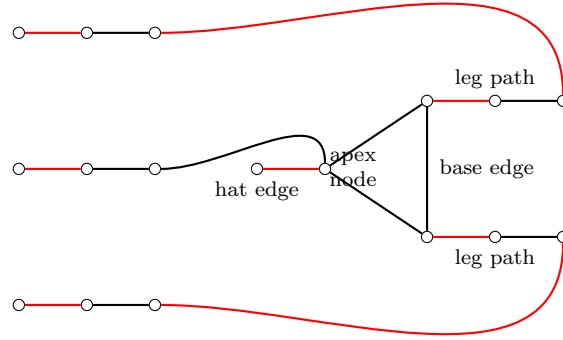
Edge-block. For a pair of color classes V_i and V_j , an *edge-block for $\{i, j\}$* consists of three length-2 *edge decider paths for $\{i, j\}$* : a *top*, a *middle*, and a *bottom* edge decider path (initially full of tokens), and an edge-gadget for each edge between V_i and V_j .

An *edge-gadget for uv* consists of a *central cone for uv* (a triangle) with an *apex node for uv* and two *base nodes for uv* (one *base node (u, uv)* and one *base node (v, uv)*), a *hat edge for uv* incident to the apex node (initially containing a token), and two length-2 *leg paths for uv* : one *leg path (u, uv)* incident to base node (u, uv) and one *leg path (v, uv)* incident to base node (v, uv) . The edge between the base nodes is called the *base edge for uv* and initially contains a token.

For a vertex u of an edge $uv \in E(G)$, the endpoint of the leg path (u, uv) that is not base node (u, uv) is called the *foot node (u, uv)* and the edge of the leg path (u, uv) that is incident to foot node (u, uv) is the *calf edge (u, uv)* while the other edge of the same leg path is the *thigh edge (u, uv)* . The *calf edges* of each edge-gadget initially contain tokens.



(a) An edge-gadget for uv (showing the initial placement of tokens) with the three edge decider paths on the left. The middle decider connects to the apex node, while top and bottom deciders connect to foot node (u, uv) and foot node (v, uv) . Initial token placement is shown in red.



(b) Possible token configuration after receiving three tokens from edge decider paths, minimizing token slides.

Figure 5.7: Edge-gadget structure and token flow.

The middle edge decider path connects from one of its endpoints (the *middle edge decider node for $\{i, j\}$*) via an edge to the apex node of every edge-gadget in the edge-block. The top edge decider path connects from one of its endpoints (the *top edge decider*

node for $\{i, j\}$) via an edge to foot node (u, uv) for every edge uv between V_i and V_j where $u \in V_i$. The bottom edge decider path connects from one of its endpoints (the *bottom edge decider node* for $\{i, j\}$) via an edge to foot node (v, uv) for every edge uv between V_i and V_j where $v \in V_j$. See Figure 5.7a.

When receiving a token from each of the edge decider paths, an edge-gadget in a solution that minimizes token slides can either (i) send all three tokens out, or (ii) keep the one from the middle edge decider path on one of its thigh edges, place the token initially on its base edge on another one of its thigh edges, and send out only two tokens from its calf edges (see Figure 5.7b). If the edge decider paths distribute their three tokens across multiple edge-gadgets within an edge-block (rather than concentrating them in one), then at least three tokens must be sent out from the edge-block, since each edge-gadget is already at maximum token capacity and cannot accommodate additional tokens (if it must accept one additional token, it must send out two of the tokens initially on its edges).

Chains and connections. The edge-gadgets and vertex-gadgets are connected via *chains*, which are paths of $r = 8\kappa^2$ *chain nodes* each adjacent to a pendant node (through an edge initially containing a token). These chains function as “toll roads”: sending a token to a chain requires sliding the token through all its edges, which is extremely expensive.

The edge-gadget for each edge $uv \in E_{i,j}$ (where $u \in V_i$, $v \in V_j$) connects to its corresponding vertex-gadgets via two dedicated chains:

- foot node (u, uv) connects via a *chain* (u, uv) to switch node (u, j) in the vertex-gadget for u .
- foot node (v, uv) connects via a *chain* (v, uv) to switch node (v, i) in the vertex-gadget for v .

We refer to the edge of chain (u, uv) for an edge $uv \in E(G)$ that is incident to a switch node by *destination edge* (u, uv) . See Figure 5.8.

The restricted budget and the length of the chains will force each edge-block to send exactly two tokens to the chains, which is the minimum number of tokens that must be sent out of an edge-gadget. The budget will also prevent any other token movement through chains. The reduction exploits a delicate balance: edge-blocks have at least two excess tokens that must evacuate to vertex-blocks, while vertex-blocks need incoming tokens to arrive at the chain destination edges incident to the switch nodes of exactly one vertex-gadget to create the augmenting paths required for efficient internal token movement inside a vertex-block.

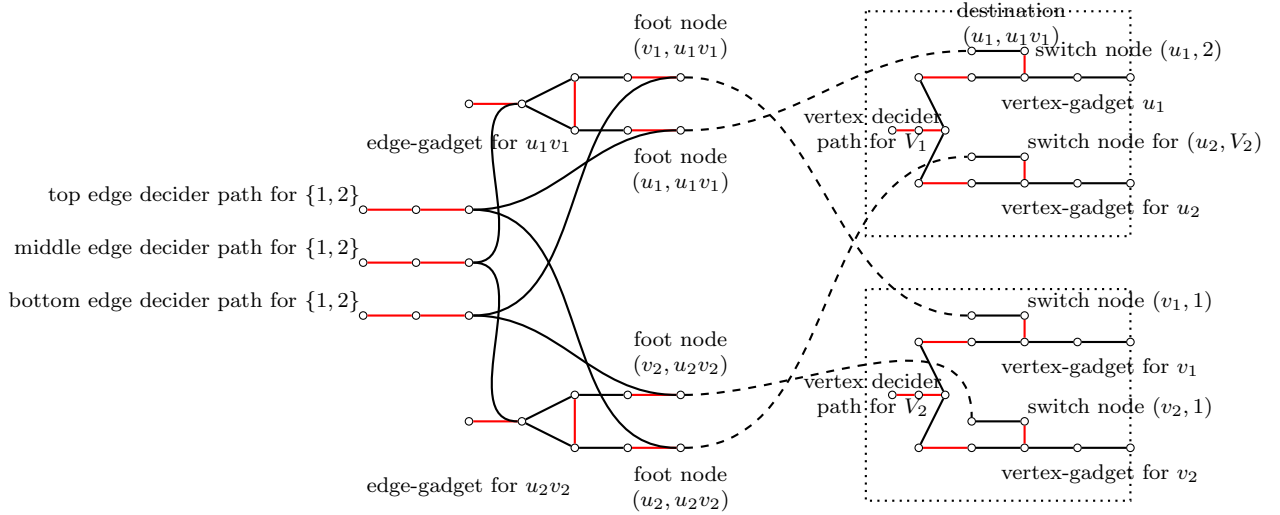


Figure 5.8: Overview of the connections between an edge-block for $\{1, 2\}$ and vertex-blocks for 1 and 2. Color class V_1 has vertices u_1 and u_2 and color class V_2 has vertices v_1 and v_2 . The graph G has edges u_1v_1 and u_2v_2 . Chains are shown as dashed lines for simplicity, with the destination edge of each chain connecting to the corresponding switch node.

Thus, the construction ensures that tokens can only flow efficiently if the selected edge-gadgets in each edge-block correspond to edges forming a multicolored clique. The chains' adjacency patterns create this verification mechanism: tokens from an edge-gadget representing edge uv can only reach (by sliding once through the chains) the switch-gadgets corresponding to the color classes of u and v in the respective vertex-gadgets of v and u . Thus, the required augmenting paths in all vertex-blocks can be simultaneously created if and only if the selected edges form a multicolored clique.

We recall that S consists of:

- all edges of vertex and edge decider paths,
- all hat and base edges,
- the calf edge of each leg path,
- in each switch-gadget: the leftmost edge and the edge incident to the switch node,
- all edges incident to pendant nodes of the chains.

The budget is set to:

$$b = \binom{\kappa}{2} \cdot (2r + 12) + \kappa \cdot (2(\kappa - 1) + 1)$$

This precisely allows each edge-block to send two tokens through chains (costing $2(r+2)$ slides per block) and each vertex-block to perform cascade evacuation II (costing $2(\kappa-1)+1$ slides), with a minimal remaining budget of $8\binom{\kappa}{2}$ for the necessary intermediate movements inside edge- and vertex-blocks.

The budget and the fact that the reduction runs in polynomial time with respect to $|V(G)|$ ensures that we obtain a valid FPT-reduction. We start by showing that H is 3-degenerate, then provide the forward direction of the proof of Theorem 50.

Claim 51. *The graph H is 3-degenerate.*

Proof of Claim 51. We prove that H is 3-degenerate by showing that every induced subgraph H' of H contains a vertex of degree at most three.

Consider any non-empty induced subgraph H' of H . First, if H' contains any pendant nodes from the chains, these have degree one in H and thus degree at most one in H' . Similarly, if H' contains any vertex select nodes or degree-1 nodes of hat edges, edge decider paths, or vertex decider paths, these have degree one in H , thus degree at most one in H' .

If H' contains no degree-1 nodes, consider the non-pendant chain nodes; these have degree at most three in H (connected to two chain neighbors and one pendant). In H' , they have degree at most two. The nodes in edge decider paths and vertex decider paths (except paths and vertex decider nodes) have degree two in H , and degree at most one in H' . The other nodes in vertex-gadgets (except switch nodes) and edge-gadgets have degree at most three in H' .

If H' contains none of the considered nodes, then it can contain only switch nodes, high-degree edge decider paths nodes, and high-degree vertex decider paths nodes. However, these nodes have degree at most 0 in H' .

Therefore, every induced subgraph H' contains a vertex of degree at most three, making H 3-degenerate. $\square$

Lemma 52. *If (G, κ) is a yes-instance of MULTICOLORED CLIQUE, then (H, S, b) is a yes-instance of MATCHING-DISCOVERY.*

Proof of Lemma 52. Let $K \subseteq V(G)$ be a (multicolored) clique in G of size κ . Let us now provide a transformation from the initial set of tokens S to a valid final configuration in b token sliding steps. We repeat the following procedure for each unordered pair of color classes V_i and V_j .

For V_i and V_j , let u be the vertex in $K \cap V_i$ and v be the vertex in $K \cap V_j$. We know that the edge uv exists since K is a clique. We perform the following three phases of token movements.

In the first phase, we fix the edge-block for $\{i, j\}$ by moving tokens from the edge decider paths for $\{i, j\}$. We do so as follows:

- we slide the token incident to the top edge decider node for $\{i, j\}$ to the edge incident to foot node (u, uv) , using one token slide,
- we slide the token incident to the bottom edge decider node for $\{i, j\}$ to the edge incident to foot node (v, uv) , using one token slide,
- we slide the token incident to the middle edge decider node for $\{i, j\}$ to thigh edge (u, uv) and the token on the base edge for uv to thigh edge (v, uv) , using four token slides, and
- we slide the token on calf edge (u, uv) to the adjacent edge belonging to chain (u, uv) and the token on calf edge (v, uv) to the adjacent edge belonging to chain (v, uv) , using two token slides.

This phase uses eight token slides per edge-block, totaling $8\binom{\kappa}{2}$ token slides.

In the second phase, tokens traverse the chains to reach the vertex-blocks. It requires $r + 1$ token slides to traverse a chain starting at an edge inside the chain incident to a foot node and ending at a destination edge incident to a switch node. Since we have $2\binom{\kappa}{2}$ tokens already on the chains, we use $\binom{\kappa}{2}(2r + 2)$ token slides.

The third phase involves adjusting the switch-gadgets to create augmenting paths. We slide the token initially on the edge incident to switch node (u, j) to rightmost edge for (u, j) and the token initially on the edge incident to switch node (v, i) to rightmost edge for (v, i) , using two token slides (that is, we form the alternative configuration inside the switch gadget). After performing this phase, for each unordered pair of color classes, the vertex-gadget for a vertex $w \in K$ has an augmenting path between the vertex select node for w and the vertex decider node for i where $w \in V_i$. This phase needs $2\binom{\kappa}{2}$ token slides.

Finally, for each color class V_i , we evacuate a token from the vertex decider path for i . Let v be a vertex in $K \cap V_i$. The augmenting path created by the alternative configurations in the vertex-gadget for v allows us to slide the token on the edge incident to the decider node for i to the edge between the vertex decider node for i and the vertex node for v , using one token slide. This triggers cascade evacuation II, where each token in the vertex-gadget

for v shifts one position right along the augmenting path, requiring $2(\kappa - 1)$ token slides. This phase uses $\kappa \cdot (2(\kappa - 1) + 1)$ token slides in total.

The total number of slides is therefore: $8\binom{\kappa}{2} + 2(r + 1)\binom{\kappa}{2} + 2\binom{\kappa}{2} + \kappa(2(\kappa - 1) + 1) = \binom{\kappa}{2}(2r + 12) + \kappa(2(\kappa - 1) + 1) = b$. $\square$

Before we present the proof for the backward direction, we state some useful properties of a *canonical solution* $(T, \vec{T})$ for the MATCHING-DISCOVERY instance (H, S, b) that uses the minimum number of token slides.

Claim 53. *In $(T, \vec{T})$, all the edges incident to the pendant nodes of the chains remain in T .*

Proof. Each pendant node has degree one and is initially matched via its unique edge. This edge should be removed from the matching if and only if one of the edges adjacent to it is in the matching. However, any such alternative matching decreases the total number of matched edges in the chains. Moreover, since the chain endpoints are already unsaturated, changing the chain configuration does not provide any benefit to the rest of the graph and only sends tokens out of the chains. $\square$

Claim 54. *The number of times we can move tokens between blocks through chains is at most $2\binom{\kappa}{2}$.*

Proof. If a token moves from one block to another through a chain, it must slide at least $r + 1$ steps since, by Claim 53, the chain has r nodes incident to tokens that remain fixed. If $2\binom{\kappa}{2} + 1$ tokens traverse chains, then they will cost at least $(2\binom{\kappa}{2} + 1)(r + 1) > \binom{\kappa}{2}(2r + 12) + \kappa(2(\kappa - 1) + 1) = b$ since $r = 8\kappa^2$. $\square$

We continue by providing an exact characterization of token movement in $\vec{T}$ from edge-blocks.

Claim 55. *In $\vec{T}$, exactly two tokens must leave the edge-block for $\{i, j\}$ (where V_i and V_j are an unordered pair of color classes) through chains. Each of those tokens slides toward an edge incident to a node in the vertex-block for i or the vertex-block for j .*

Proof. The edge-block for $\{i, j\}$ contains $9 + 8|E_{i,j}|$ nodes (nine nodes from the edge decider paths, and eight nodes inside each edge-gadget) and $6 + 4|E_{i,j}|$ tokens in S (six tokens on the edge decider paths, and four on edges of each edge-gadget). Since the block can accommodate a matching of size at most $\lfloor (9 + 8|E_{i,j}|)/2 \rfloor = 4 + 4|E_{i,j}|$, at least two tokens

must leave through chains. A token from an edge-block reaches a corresponding vertex-block by traversing one chain, but reaches another edge-block by traversing at least two chains. Since there are $\binom{\kappa}{2}$ edge-blocks and we can traverse chains at most $2\binom{\kappa}{2}$ times by Claim 54, exactly two tokens leave each edge-block and each of those tokens traverses once exactly one chain. Moreover, since each chain from an edge-gadget for uv (where $u \in V_i, v \in V_j$) connects to the vertex-block for i or the vertex-block for j , the tokens that leave the edge-block for $\{i, j\}$ slide each toward an edge incident to a node in one of these vertex-blocks. $\square$

Next, we establish the minimum cost for reconfiguring tokens within each edge-block.

Claim 56. *In $\vec{T}$, there are at least eight token slides whose source edges are inside the edge-block for $\{i, j\}$ (where V_i and V_j are an unordered pair of color classes). Additionally, the two tokens that leave the edge-block for $\{i, j\}$ leave from the same edge-gadget (say, the edge-gadget for uv).*

Finally, if only eight token slides whose source edges are inside the edge-block for $\{i, j\}$ are in $\vec{T}$, then of the two tokens that leave the edge-block for $\{i, j\}$, one slides through chain (u, uv) towards an edge incident to switch node (u, j) and the other slides through chain (v, uv) towards an edge incident to switch node (v, i) .

Proof of Claim 56. From Claim 55, the edge-block for $\{i, j\}$ must accommodate exactly $4 + 4|E_{i,j}|$ tokens in T . Additionally, it must slide one token from each of the three edge decider paths for $\{i, j\}$ at least once.

If these three tokens slide in the direction of multiple edge-gadgets in the edge-block for $\{i, j\}$, at least three tokens must leave these edge-gadgets (since no edge-gadget can accommodate an extra token). Given Claim 55, exactly two tokens leave each edge-block. Thus, the three tokens must slide toward a single edge-gadget (w.l.o.g. the edge-gadget for uv) in the edge-block for $\{i, j\}$ which can send out exactly two tokens.

This is achievable in a canonical solution only if the edge-gadget uv (where $u \in V_i, v \in V_j$) changes its configuration, while all other edge-gadgets in the edge-block for $\{i, j\}$ retain their initial configurations. Specifically, the edge-gadget for uv and its connections to the edge decider paths for $\{i, j\}$ must have the following configuration in T (see Figure 5.7b):

- the edge {top edge decider node for $\{i, j\}$, foot node (u, uv) } has a token,
- the edge {bottom edge decider node for $\{i, j\}$, foot node (v, uv) } has a token,

- the edge {middle edge decider node for $\{i, j\}$, apex node for uv } does not have a token,
- the hat edge for uv maintains its token,
- all edges of the central cone for uv do not have tokens, and
- the thigh edges have tokens but the calf edges do not.

The only sequence that achieves this configuration in the minimum number of token slides requires:

- one token slide to slide the token incident to the top decider node for $\{i, j\}$ to the edge this node shares with foot node (u, uv) ,
- one token slide to slide the token incident to the bottom decider node for $\{i, j\}$ to the edge this node shares with foot node (v, uv) ,
- three token slides to slide the token incident to the middle decider node for $\{i, j\}$ to one of thigh edge (u, uv) or thigh edge (v, uv) ,
- one token slide to slide the token on the base edge for uv to the token-free thigh edge in the same gadget,
- two token slides to slide each of the tokens on the calf edges in the gadget one token sliding step toward the corresponding chain.

This totals at least eight token slides within the edge-block. Additionally, since $\vec{T}$ is the sequence of a canonical solution that uses the minimum number of token slides and from Claim 55, we know that of the two tokens that leave the edge-block for $\{i, j\}$ towards the chains, one slides through chain (u, uv) towards the destination edge incident to switch node (u, j) and the other slides through chain (v, uv) towards the destination edge incident to switch node (v, i) . $\square$

We are now ready to present the backward direction.

Lemma 57. *If (H, S, b) is a yes-instance of MATCHING-DISCOVERY, then (G, κ) is a yes-instance of MULTICOLORED CLIQUE.*

Proof of Lemma 57. Let $(T, \vec{T})$ be a canonical solution for the instance (H, S, b) of MATCHING-DISCOVERY.

First, we analyze the token movements in vertex-blocks. A vertex decider path for i has two tokens and at least one must slide in $\vec{T}$. The token cannot slide through a chain, since by Claim 55, for each edge-block, exactly two tokens must slide through chains, and the number of times we can move tokens between blocks through chains is at most $2\binom{\kappa}{2}$ by Claim 54. Thus, in $\vec{T}$, the token slides within the vertex-block for i .

With an augmenting path in some vertex-gadget of the vertex-block for i (created when all switch-gadgets in the vertex-gadget are in alternative configuration), cascade evacuation II uses a total of $2(\kappa - 1) + 1$ token slides. Without an augmenting path in some vertex-gadget of the vertex-block for i , we will need more token slides.

Given that we need at least $8\binom{\kappa}{2}$ token slides inside edge-blocks (by Claim 56), $\binom{\kappa}{2}2(r + 1)$ token slides to move tokens through chains (by Claim 55), and at least $2\binom{\kappa}{2}$ token slides for the tokens that arrive at destination edges (we have $2\binom{\kappa}{2}$ tokens that arrive at destination edges), we have only $\kappa \cdot (2(\kappa - 1) + 1)$ token slides remaining for vertex-blocks. Note that the tokens that arrive at destination edges can in a canonical solution only either slide toward the adjacent edge incident to a switch node or trigger the token on that edge to slide toward the vertex-gadget it initially belongs to, but not slide back to an edge in a chain.

Thus, from the remaining budget, we know that each vertex-block must use cascade evacuation II. This requires at least one vertex-gadget per vertex-block to have all its switch-gadgets in the alternative configuration. To achieve this without extra token slides, each such vertex-gadget must receive exactly one token at each of the destination edges incident to its switch nodes, forcing the token initially on an edge incident to a switch node in a switch-gadget of the gadget to slide toward the rightmost edge of the same switch-gadget. Notice that if the received token at the destination edge slides instead, it will require more than one token slide, which is impossible. The number of described token slides is equal to b , thus exactly one vertex-gadget per vertex-block has all its switch-gadgets in the alternative configuration.

From Claim 56, an edge-block for $\{i, j\}$ sends exactly two tokens through the chains from a single edge-gadget. Let uv be the edge whose edge-gadget sends out tokens in the edge-block for $\{i, j\}$. From Claim 56, of the tokens that the edge-gadget for uv sends out through the chains, one of these tokens slides through chain (u, uv) towards destination edge (u, j) and the other slides through chain (v, uv) towards destination edge (v, i) . For the vertex-gadget for u to receive a token on each destination edge incident to one of its $\kappa - 1$ switch nodes, it must be that each such edge is a destination of a chain-traversing

token initially on an edge-block for $j \neq i$. The same holds for the vertex-gadget for v . Since this holds for every single edge-gadget selected in an edge-block and consequently for every single vertex-gadget selected in a vertex-block, the vertices represented by the selected vertex-gadgets and the edges represented by the selected edge-gadgets must form a multicolored clique in G . $\square$

5.2 Vertex/Edge Cut-Discovery

In this section, we focus on VERTEX/EDGE CUT-DISCOVERY. We show in Section 5.2.1 that the problems are NP-hard and $W[1]$ -hard with respect to parameter b through a reduction from CLIQUE (which we define in the same section).

In Section 5.2.2, we show that the problems are fixed-parameter tractable when parameterized by k by reducing them to weighted rainbow variants of the classical VERTEX/EDGE CUT problems through color-coding, where colors encode token-to-vertices assignments and weights capture token sliding costs. The algorithm combines this reduction with a treewidth reduction theorem from Marx et al. [143] and Courcelle's theorem on bounded treewidth graphs, enabling us to find minimum-weight cuts satisfying rainbow constraints in fixed-parameter tractable time.

In Section 5.2.3, we show that the problems do not admit polynomial kernels with respect to the same parameter unless $\text{NP} \subseteq \text{coNP}/\text{P}$ through an or-cross-composition from RAINBOW MATCHING (which we define in the same section).

5.2.1 Hardness of Vertex/Edge Cut-Discovery

In this section, we show that VERTEX/EDGE CUT-DISCOVERY is NP-hard and $W[1]$ -hard when parameterized by b even on 2-degenerate bipartite graphs and we start by proving the following theorem.

Theorem 58. VERTEX CUT-DISCOVERY is NP-hard and $W[1]$ -hard when parameterized by b on 2-degenerate bipartite graphs.

Recall from Section 3.3 of the preliminaries, in a graph G , a *vertex cut* between two vertices s and t (equivalently, an s - t vertex cut) is a set of vertices $C \subseteq V(G)$ such that every path from s to t in G contains a vertex in C . Likewise, an *edge cut* in a graph G between two of its vertices s and t (equivalently, an s - t edge cut) is a set of edges $C \subseteq E(G)$ such that every path from s to t in G contains an edge of C .

To prove Theorem 58, we reduce from the CLIQUE problem. In the CLIQUE problem, we are given a graph G and an integer κ , and the goal is to decide whether G contains a clique of size κ . The CLIQUE problem is NP-hard [126] and W[1]-hard when parameterized by κ [67].

Let (G, κ) be an instance of CLIQUE. We may assume that $\kappa \geq 4$, hence $\binom{\kappa}{2} > \kappa$. For clarity, we refer to *vertices* in the instance of CLIQUE and *nodes* in our construction.

We construct a graph H with special nodes s and t that we want to separate. See Figure 5.9. We make s adjacent to κ pendant *reservoir nodes* (nodes of degree one).

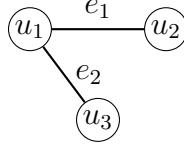
For each vertex $u \in V(G)$, we create a *vertex node for u* adjacent to s . For each edge $uv \in E(G)$, we create an *edge node for uv* adjacent to t . For each vertex u and edge uv in G , we connect the vertex node for u and the edge node for uv by a new path of length four (with three internal nodes).

Finally, we add $\binom{\kappa}{2}$ other *verifier paths* of length two between s and t . Each such path has an internal *verifier node* that must have a token to separate s from t .

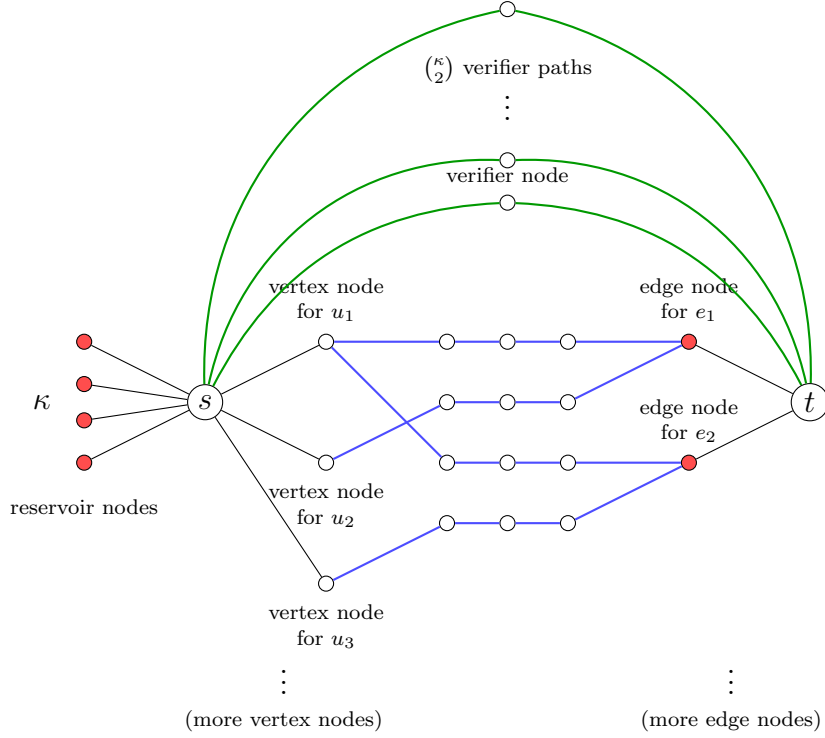
Initially, we place tokens on all the reservoir and edge nodes and set the budget to $b = 2\kappa + 2\binom{\kappa}{2}$. The budget and the fact that the reduction runs in polynomial time with respect to $|V(G)|$ ensures that we obtain a valid FPT-reduction. The key insight is that to separate s and t , we must block all $\binom{\kappa}{2}$ verifier paths (by placing tokens on their verifier nodes) and all then unblocked paths (between s and t) that go through vertex and edge nodes. Since only tokens from reservoir nodes can reach the vertex nodes in two token slides (via s) and tokens from edge nodes can reach the verifier nodes in two token slides (via t), the budget and the construction will force us to use exactly these movements. Additionally, since $\binom{\kappa}{2}$ tokens must move from edge nodes to verifier nodes, $\binom{\kappa}{2}$ edge nodes will be without tokens in T . The unblocked paths (between s and t) through these token-free edge nodes must be blocked by tokens on vertex nodes. Since we can only move κ tokens from reservoir to vertex nodes, these κ vertex nodes must correspond to vertices that form a clique in G to ensure that all paths (between s and t) are blocked.

Lemma 59. *H is a 2-degenerate bipartite graph.*

Proof. First, we show that H is bipartite. To do that, we need to show that we can partition the vertices in $V(H)$ into two sets such that no two vertices of a set are adjacent in H . We construct a bipartition (L, R) of $V(H)$ as follows. Place in L the nodes s and t , and the first and last internal nodes of each path connecting vertex nodes to edge nodes. Place in R all the vertex nodes and the edge nodes, all the reservoir and verifier nodes, and the remaining internal nodes of the paths connecting vertex nodes to edge nodes. Since all edges in G go between L and R , the graph H is bipartite.



(a) A part of a CLIQUE instance showing vertices u_1, u_2, u_3 and edges $e_1 = u_1u_2, e_2 = u_1u_3$.



(b) The reduced graph H . Blue paths of length four connect vertex nodes to edge nodes when the corresponding vertex is incident to the edge in G . For example, since $e_1 = u_1u_2$, there are paths from both the vertex node for u_1 and the vertex node for u_2 to the edge node for e_1 . Initially, tokens (red) are placed on all reservoir nodes and all edge nodes.

Figure 5.9: Reduction from CLIQUE to VERTEX CUT DISCOVERY.

To show that H is 2-degenerate, we prove that every non-empty subgraph of H contains a vertex of degree at most two. Consider any non-empty subgraph $H' \subseteq H$.

If H' contains any reservoir node, such a node has degree one in H , thus degree at most one in H' . If H' contains any internal node from the paths connecting vertex nodes

to edge nodes, such a node has degree exactly two in H , thus degree at most two in H' . If H' contains any verifier node, such a node has degree exactly two in H , thus degree at most two in H' . If H' contains none of the above, then H' either contains a vertex or an edge node that has degree at most one in H' or only the vertices s and t , that then have degree 0 in H' .

Therefore, every non-empty subgraph of H contains a vertex of degree at most two, proving that H is 2-degenerate. $\square$

Lemma 60. *If (G, κ) is a yes-instance of CLIQUE, then (H, s, t, S, b) is a yes-instance of VERTEX CUT-DISCOVERY.*

Proof of Lemma 60. Let $K \subseteq V(G)$ be a clique of size κ in G . We show how to slide tokens to create a vertex cut that separates s and t using $b = 2\kappa + 2\binom{\kappa}{2}$ token slides.

Take the κ tokens on the reservoir nodes and slide them through s to the vertex nodes for vertices in K . This costs two token slides per token, totaling 2κ slides.

Since K is a clique, there are exactly $\binom{\kappa}{2}$ edges in G with both endpoints in K . For each such edge e , the corresponding edge node is now “freed” because (both) its paths to s are blocked by tokens on vertex nodes. We take the tokens on these $\binom{\kappa}{2}$ edge nodes and slide them through t to the verifier nodes. This costs two token slides per token, totaling $2\binom{\kappa}{2}$ slides.

The resulting configuration has tokens on all vertex nodes for vertices in K , all edge nodes for edges with at least one endpoint not in K , and all verifier nodes. This forms a vertex cut that separates s and t and only $2\kappa + 2\binom{\kappa}{2} = b$ token slides were used to achieve this cut. $\square$

Lemma 61. *If (H, s, t, S, b) is a yes-instance of VERTEX CUT DISCOVERY, then (G, κ) is a yes-instance of CLIQUE.*

Proof. Let T be a vertex cut in H separating s and t , obtainable from S using at most $b = 2\kappa + 2\binom{\kappa}{2}$ token slides.

First, observe that any vertex cut must contain all $\binom{\kappa}{2}$ verifier nodes of the verifier paths, since each such path must be blocked. Since moving a token to a verifier node requires at least two token slides, we need at least $2\binom{\kappa}{2}$ token slides just for blocking these paths.

With the remaining budget of 2κ token slides, we can move at most κ tokens from the reservoir nodes. Let r be the number of tokens moved from the reservoir nodes to

the verifier nodes, and $x = \kappa - r$ be the number of tokens moved from the reservoir nodes to vertex nodes. The x tokens on vertex nodes can block paths to at most $\binom{x}{2}$ edge nodes (those corresponding to edges between the selected vertices). Since all paths must be blocked, we need $\binom{x}{2} \geq \binom{\kappa}{2} - r$. For $\kappa \geq 4$, this inequality only holds when $r = 0$. Therefore, all κ tokens from reservoir nodes move to vertex nodes, and all $\binom{\kappa}{2}$ tokens that move to verifier nodes come from edge nodes.

An edge node for edge uv can have its token moved only if (both) paths from it to s are already blocked, which happens only if both the vertex node for u and the vertex node for v have tokens. Thus, with only κ tokens on vertex nodes, the $\binom{\kappa}{2}$ edge nodes that give up their tokens correspond to edges in G that form a clique of size κ . $\square$

The proof for EDGE CUT-DISCOVERY follows naturally from our construction for VERTEX CUT-DISCOVERY. Since in our construction, filling nodes is equivalent to filling the unique edge they share with s or t , we can adapt the same construction by placing tokens on edges instead of vertices.

Corollary 62. *EDGE CUT-DISCOVERY is NP-hard and $W[1]$ -hard when parameterized by b on 2-degenerate bipartite graphs.*

Proof of Corollary 62. We use the same graph H but place tokens on the edges instead of on the nodes. Specifically, we place tokens on the edges between the reservoir nodes and s , and on the edges between the edge nodes and t . We set the budget to $b = \kappa + \binom{\kappa}{2}$. The reduction works similarly, with tokens sliding along edges instead of nodes. $\square$

5.2.2 Fixed-Parameter Tractability of Vertex/Edge Cut-Discovery for Parameter k

In this section, we establish that VERTEX CUT-DISCOVERY and EDGE CUT-DISCOVERY are fixed-parameter tractable when parameterized by the number of tokens k . Our approach unfolds in three key steps. First, we introduce weighted rainbow variants of the cut problems (that is, VERTEX CUT and EDGE CUT), where we seek cuts whose elements have distinct colors and whose total weight is bounded. Then, we show that the rainbow constraint, combined with bounded treewidth techniques from Marx [143], allows us to apply powerful algorithmic machinery (specifically, Courcelle's theorem on graphs of bounded treewidth), which gives us fixed-parameter tractable algorithms for the weighted rainbow variants.

Finally, we show that, using the color-coding technique of Alon et al. [7], any instance of a CUT DISCOVERY problem can be transformed into a weighted rainbow variant instance where the colors encode which tokens should move where and the weights capture the sliding costs. When coloring is done correctly (which happens with probability at least e^{-k}), a solution to the weighted rainbow variant instance directly translates into a solution for the original discovery problem instance. Repeating the randomized process sufficiently many times gives us a fixed-parameter tractable algorithm for the original discovery problem.

We begin by introducing the weighted rainbow variants of each of the VERTEX CUT and EDGE CUT problems, which will serve as the key intermediate problems in our algorithm.

WEIGHTED RAINBOW VERTEX CUT

Input: A vertex-colored graph $(G, \mathcal{C})$ where every vertex has one color, two vertices $s, t \in V(G)$, vertex weights $\omega : V(G) \rightarrow \mathbb{R}_+$, and integers κ and ℓ .

Output: YES if there exists a $C \subseteq V(G) \setminus \{s, t\}$ such that every path from s to t in G has a vertex in C , $|C| \leq \kappa$, all vertices in C have different colors, and $\sum_{v \in C} \omega(v) \leq \ell$. NO otherwise.

WEIGHTED RAINBOW EDGE CUT

Input: An edge-colored graph $(G, \mathcal{C})$ where every vertex has one color, two vertices $s, t \in V(G)$, edge weights $\omega : E(G) \rightarrow \mathbb{R}_+$, and integers κ and ℓ .

Output: YES if there exists $C \subseteq E(G)$ such that removing C disconnects s from t , $|C| \leq \kappa$, all edges in C have different colors, and $\sum_{e \in C} \omega(e) \leq \ell$. NO otherwise.

The key insight is that the discovery problem can be reduced to the weighted rainbow problem using the color-coding technique of Alon et al. [7]. We illustrate this connection for vertex cuts (the edge cut case is analogous).

For (G, S, s, t, b) , an instance of VERTEX CUT-DISCOVERY, we construct a WEIGHTED RAINBOW VERTEX CUT instance $(H, \mathcal{C}, s, t, \omega, \kappa, \ell)$ as follows.

First, we let $H = G$ and $\kappa = k$. For clarity, we refer to *vertices* in the instance of VERTEX CUT-DISCOVERY and *nodes* in our construction. For each vertex $u \in V(G)$, we let u be the node representing the vertex u in H . We define U as the set of nodes in H corresponding to the vertices in S . Let $\pi : [\kappa] \rightarrow U$ be a bijection. We fix κ colors $\{C_1, \dots, C_\kappa\}$, assign to each node outside U a color chosen uniformly at random from these colors, and the color $C_{\pi(v)}$ to each node $v \in U$, thus producing a coloring $\mathcal{C}$. We give each

node v in H a weight $\omega(v)$ that represents the minimum number of token slides required to move a token from a node in U having the same color as v to v . Note that the nodes in U have zero weight. Finally, we set $\ell = b$.

Lemma 63. *If $(H, \mathcal{C}, s, t, \omega, \kappa, \ell)$ is a yes-instance of WEIGHTED RAINBOW VERTEX CUT, then (G, S, s, t, b) is a yes-instance of VERTEX CUT-DISCOVERY.*

Proof. Suppose C is a vertex cut of weight at most ℓ whose nodes having different colors in the colored instance. Since C forms a valid vertex cut and is formed of nodes having different colors, we can slide tokens to vertices represented by the nodes in C by matching colors: for each node $v \in C$, we move the token from the vertex $\pi(i)$ where i the index of the color assigned to the node v in H , to the vertex v . The total number of token slides needed equals $\sum_{v \in C} \omega(v) \leq \ell$. This makes the cut in (G, S, s, t, b) achievable within budget. $\square$

Lemma 64. *If (G, S, s, t, b) is a yes-instance of VERTEX CUT-DISCOVERY, then the probability that $(G, \mathcal{C}, s, t, \omega, \kappa, \ell)$ is a yes-instance of WEIGHTED RAINBOW VERTEX CUT is at least e^{-k} .*

Proof. Let $(T, \vec{T})$ be a solution to the (G, S, s, t, b) instance, and let C be the nodes that represent the vertices in T . For C to be a valid solution for the WEIGHTED RAINBOW VERTEX CUT instance, the vertices of C must receive distinct colors. Since U uses all k colors uniquely and the nodes outside U are colored randomly, the probability that the nodes of C have distinct colors is at least $k!/k^k \geq e^{-k}$. $\square$

These lemmas establish that solving the weighted rainbow problem yields solutions to the discovery problem with some success probability. Repeating the random coloring e^k times increases the success probability to a constant.

Furthermore, we can eliminate randomization using (n, k, k) -perfect hash families, which were defined in Section 2.1. We recall that an (n, k, k) -perfect hash family is a collection $\mathcal{F}$ of functions from $[n]$ to $[k]$ such that for every k -element subset $S \subseteq [n]$, there exists a function $f \in \mathcal{F}$ that is injective on S . Modern constructions achieve families of size $e^k k^{O(\log k)} \log n$ that can be generated in polynomial time (for a fixed k) [149].

Rather than random coloring, we can construct a (n, k, k) -perfect hash family $\mathcal{F}$, which guarantees that for any k -subset outside U , at least one function $f \in \mathcal{F}$ is such that every element of this subset is hashed to a distinct value in $[k]$. This gives us a deterministic fixed-parameter tractable algorithm reduction.

Theorem 65. VERTEX/EDGE CUT-DISCOVERY admits a fixed-parameter tractable algorithm when parameterized by k if and only if WEIGHTED RAINBOW VERTEX/EDGE CUT admits a fixed-parameter tractable algorithm when parameterized by κ .

To establish the tractability of the weighted rainbow problems, we need the following key result that will reduce G to a graph of treewidth bounded by some function of κ and allow us to apply Courcelle's Theorem (Theorem 67) to find a minimum weight cut.

Theorem 66 (Theorem 2.15 of [143]). *Given a graph G , a set $D \subseteq V(G)$, and a non-negative integer κ , let K be the set of all vertices of G participating in a minimal cut of size at most κ between some $s, t \in D$. There exists a linear-time algorithm that computes a graph H with the following properties:*

- $K \cup D \subseteq V(H)$,
- for all $s, t \in D$, a set $C \subseteq V(H)$ with $|C| \leq \kappa$ is a minimal cut between s and t in H if and only if $C \subseteq K \cup D$ and C is a minimal cut between s and t in G , and
- the treewidth of H is at most $h(\kappa, |D|)$ for some function h ; and $H[K \cup D]$ is isomorphic to $G[K \cup D]$.

Theorem 67 ([10, 55]). *Let ϕ be an MSO-formula and $\omega_1, \dots, \omega_m$ be weight functions on the vertices or edges of a graph. For every fixed tw and m , there exists a linear-time algorithm that, given a graph G of treewidth at most tw , computes an optimal solution to the problem of finding a set O satisfying ϕ that minimizes $\sum_{i=1}^m c_i \cdot \omega_i(O)$ for given constants $c_1, \dots, c_m$.*

We are now ready to prove the fixed-parameter tractability of WEIGHTED RAINBOW VERTEX/EDGE CUT when parameterized by κ .

Theorem 68. WEIGHTED RAINBOW VERTEX/EDGE CUT is fixed-parameter tractable when parameterized by κ .

Proof. We present a fixed-parameter tractable algorithm for the WEIGHTED RAINBOW VERTEX CUT problem. Let $(G, \mathcal{C}, s, t, \omega, \kappa, \ell)$ be an instance of WEIGHTED RAINBOW VERTEX CUT.

As argued in Section 3.3, the tractability of WEIGHTED RAINBOW EDGE CUT follows by considering the vertex cut version in the line graph of $(G, \mathcal{C})$ (colors inherited from edges to their representative vertices) between the vertices u , which is a new vertex added

to the line graph and made adjacent to all vertices representing edges incident to s in G , and v , which is a new vertex added to the line graph and made adjacent to all vertices representing edges incident to t in G .

We apply Theorem 66 to G with $D = \{s, t\}$ to obtain a subgraph H of G . The colors and weights are inherited from G to H . Since all minimal cuts of size at most κ between s and t are preserved in H (by Property 2 of the theorem), it suffices to search in H for a minimum weight cut between s and t of size at most κ whose vertices having distinct colors.

Since H has treewidth at most $h(\kappa, 2)$ (by Property 3), we can apply Courcelle's Theorem (Theorem 67). We formulate the existence of such a minimal cut using an MSO-formula $\phi(X)$ such that $\phi(C)$ is true for a set C of vertices if and only if:

- there is no path from s to t in $G[V(H) - C]$ which we can write as $\exists U \subseteq V(H) : [U(s) \wedge \neg U(t) \wedge \forall u \in V(H) : [U(u) \rightarrow \neg X(u)] \wedge \forall u, v \in V(H) : ([U(u) \wedge \neg U(v) \wedge \neg X(v)] \rightarrow \neg E(u, v))]$ where U simply the set of all vertices reachable from s in $G[V(H) - C]$,
- $|C| \leq \kappa$ which we can write as $\exists x_1, \dots, x_\kappa \in V(H) : [\forall y \in X : (\bigvee_{i=1}^\kappa y = x_i)]$, and
- for all distinct $u, v \in C$, the color of u and the color of v are different, which we can write as $\forall u, v \in X : [u \neq v \rightarrow (\bigwedge_{i=1}^c \neg[(C_i(u) \wedge C_i(v))]]$.

All three conditions can be expressed in MSO (with a formula size that depends on κ). We can apply Theorem 67 with the weight function ω to find a minimum weight set C that satisfies ϕ in time $f(\kappa) \cdot n$, where f is a computable function depending only on κ . $\square$

Combining our reductions with the fixed-parameter tractable algorithms for weighted rainbow cuts, we obtain our main result.

Theorem 69. VERTEX CUT-DISCOVERY and EDGE CUT-DISCOVERY are fixed-parameter tractable when parameterized by k .

5.2.3 No Polynomial Kernel for Vertex/Edge Cut-Discovery with Parameter k

In this section, we show that VERTEX CUT-DISCOVERY when parameterized by k does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{P}$. We do so by providing an or-cross-composition from RAINBOW MATCHING. Thus, we prove the following theorem.

Theorem 70. *There exists an or-cross-composition from RAINBOW MATCHING to VERTEX CUT-DISCOVERY parameterized by k . Consequently, VERTEX CUT-DISCOVERY does not admit a polynomial kernel with respect to k , unless $NP \subseteq coNP/P$.*

Let $(G, \mathcal{C})$ be an edge-colored graph where each edge has one color. Recall from Section 5.1.1, we say that an edge coloring of $E(G)$ is *proper* if no two adjacent edges have the same color. A matching $\mathcal{M} \subseteq E(G)$ is said to be a *rainbow matching* if all edges in $\mathcal{M}$ have pairwise distinct colors. In the RAINBOW MATCHING problem, we are given an edge-colored graph $(G, \mathcal{C})$ where each edge has one color and an integer κ . The goal is to decide whether there exists a rainbow matching of size κ in $(G, \mathcal{C})$. The problem is NP-complete even on properly colored paths where every color appears exactly twice [137].

Also recall, from the preliminaries, that a problem A *or-cross-composes* into a problem B if there is a polynomial equivalence relation $\mathcal{R}$ on the set of instances of A and an algorithm that, given q instances belonging to the same equivalence class of $\mathcal{R}$ computes in time polynomial in the input, an instance of B that is a yes-instance of B if and only if one of the q instances of A is a yes-instance of A and such that the parameter for the instance of B is bounded above by a polynomial in the size of the maximum size instance of the q instances and $\log q$.

Let $\mathcal{R}$ be the polynomial equivalence relation where two instances of RAINBOW MATCHING are equivalent if they have the same number of vertices (which we denote by n), the same number of edges (which we denote by m), and where the value κ is the same. This relation can be decided in polynomial time and partitions RAINBOW MATCHING instances into classes based on their numbers of vertices, edges, and value of κ . For clarity, we refer to *vertices* in any instance of RAINBOW MATCHING and *nodes* in our construction. Additionally, we assume that the number of instances q of RAINBOW MATCHING is a power of two, that is, $q = 2^r$ for some integer r (if not, we can duplicate instances as needed, adding at most q additional instances), and that the q instances have the same number of vertices (which we denote by n), the same number of edges (which we denote by m), and the value κ is the same. Moreover, we denote instance $j \in [q]$ of RAINBOW MATCHING by $((G^j, \mathcal{C}^j), \kappa^j)$. The high-level idea of our construction is to create a graph H where disconnecting two special nodes requires selecting exactly one input instance to disconnect and, within that instance, choosing edges that form a rainbow matching. We achieve this through a careful design, where the final configuration encodes both the instance selection and the edge choices, with connectivity patterns that enforce the matching and rainbow constraints.

The key insight for instance selection is to exploit a limited token budget with a binary tree structure. First, for each $j \in [q]$, we will have one *instance component* in H , which

we refer to as the *instance component for j* or H^j for short. The instance component for j will have two *local special nodes for j* (namely, the *local special s node for j* and the *local special t node for j*), and will be such that none of its vertices are adjacent to any of the vertices of other components for other instances. These instance components, specifically their respective local special s nodes, will become the leaves of a *binary tree structure* (with very long $m^3 + \log q$ -length paths between its levels, to discourage traversal) rooted at a *global special s node*, and will connect, specifically through their respective local special t nodes, to a *global special t node*. See Figure 5.10.

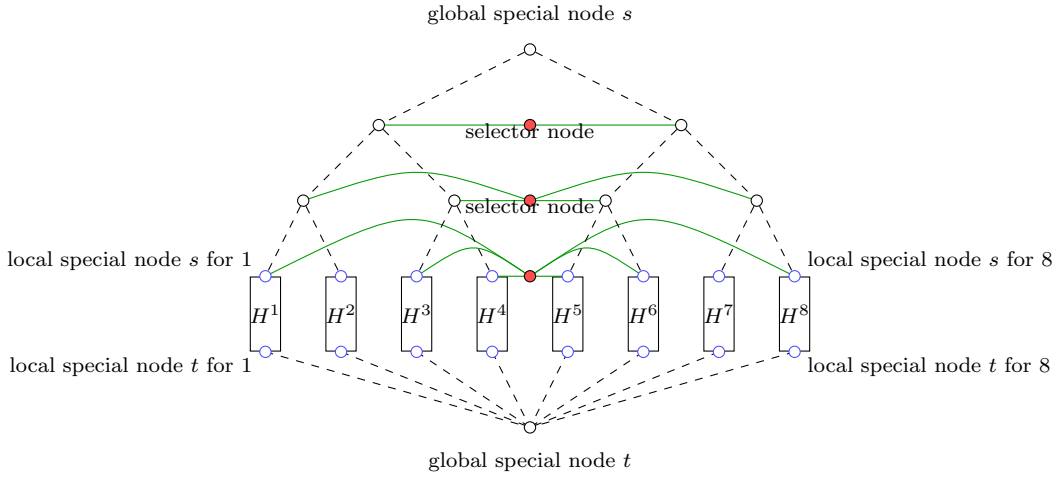


Figure 5.10: The instance selection mechanism via a binary tree structure. Eight instance components (shown as rectangles) have their local special s nodes form the leaves of a binary tree structure rooted at the global special s node. Three selector nodes, one at each depth of the tree, are each adjacent to all nodes at their respective depth level (green edges show these adjacencies). Each selector node can block one branch at its depth. With these $\log_2 8 = 3$ tokens making binary choices, exactly seven paths can be blocked, selecting precisely one instance. Each component's local special t node connects to the global special t node. Dashed lines represent paths of length $m^3 + \log t$.

In H , we must disconnect the global special s node from the global special t node, and to do so with a limited budget, we must block all paths through this binary tree

structure leading to all but one component. We provide exactly $\log q$ tokens on *selector nodes*, each adjacent to all nodes at a depth level of the binary tree structure. Crucially, these tokens cannot traverse the long paths between the binary tree structure levels within our budget, so each token must make a local binary decision at its assigned depth. At depth one, the token can block one of two branches, effectively eliminating half of the paths remaining to instance components, and thus to the global special t node. After $\log q$ such binary decisions, exactly $q - 1$ paths are blocked, leaving exactly one instance component reachable from the global special s node. We call an instance component that remains reachable from the global special s node after the tokens on selector nodes have moved to their destinations in T a *selected instance component*. This use of a binary tree with depth-restricted tokens ensures that we select precisely one instance while using a minimal number of tokens.

We are now ready to describe the construction of an instance component whose final configuration, if containing any tokens, must encode a rainbow matching through careful token placement that cuts the connectivity patterns that we create.

For each $j \in [t]$, we construct the instance component for j as follows. The component contains the two *local special nodes* for j that serve as entry and exit points for all subpaths of the paths in H going from the global special s node to the global special t node through the instance component for j . To disconnect the global special nodes, we must disconnect these local special nodes within the selected instance component. However, within our budget, no token can reach these local special nodes of instance components.

In fact, disconnecting the two local special nodes for j will occur only when, for each layer $\ell \in [\kappa - 1]$ within the instance component for j , we disconnect the *special s node for layer ℓ of component j* from the *special t node for layer ℓ of component j* . See Figure 5.11. Just like the local special nodes of instance components, the special nodes of each layer will each be unreachable by a token within our budget, so disconnecting these nodes must be accomplished by other means.

We first describe what each layer of an instance component for j consists of. In each layer $\ell \in [\kappa - 1]$, we create a set of $2m + 2$ nodes. Two of these are the *special nodes for layer ℓ in component j* . The remaining $2m$ *edge nodes* represent the edges of G^j , with each edge represented twice. We call these the *first copy node of edge e in layer ℓ of component j* (*first copy* (e, ℓ, j) , for short) and the *second copy node*, (*second copy* (e, ℓ, j) , for short). We also call the set of nodes containing for each $e \in E(G)$, the first copy node of edge e in layer ℓ of component j , the *first copy nodes in layer ℓ of component j* , and the set of nodes containing for each $e \in E(G)$, the second copy node of edge e in layer ℓ of component j , the *second copy nodes in layer ℓ of component j* .

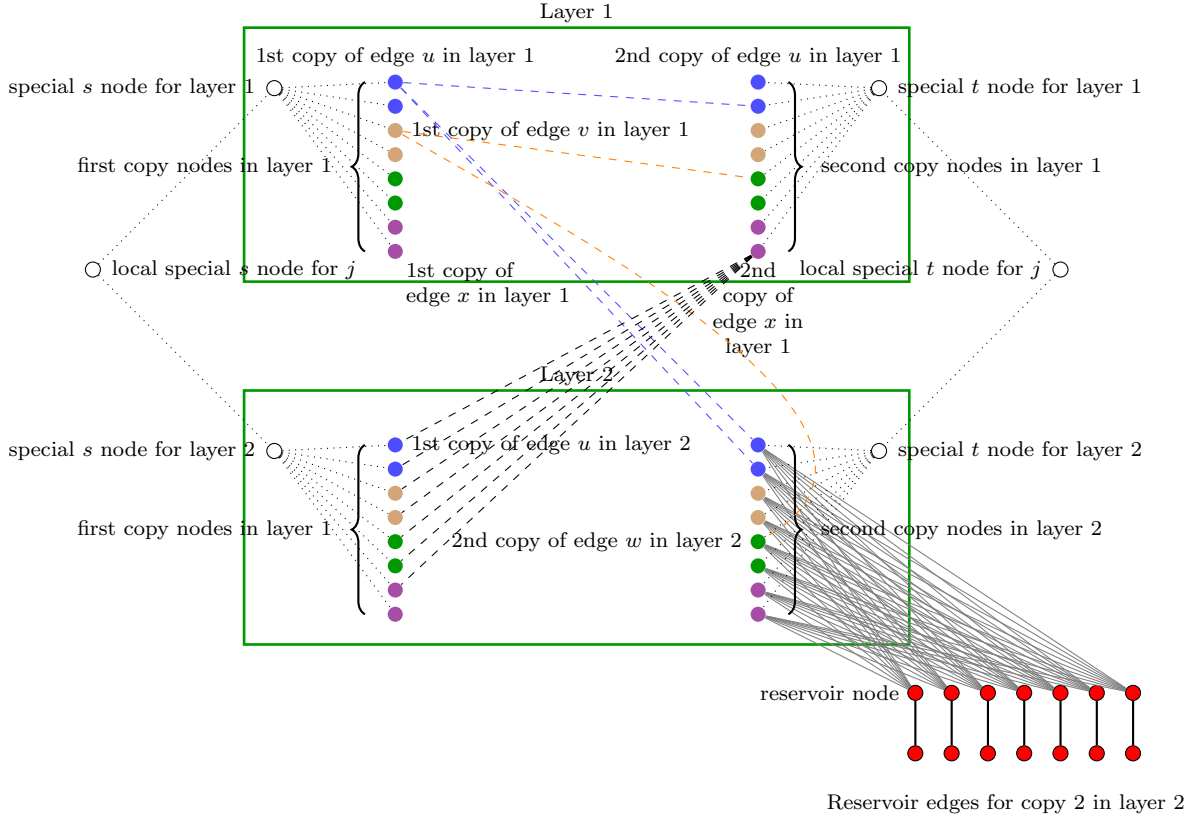


Figure 5.11: Internal structure of component H^j showing two layers. Edge nodes are colored with the color of the edges they represent. Dashed or dotted edges are paths of length $m^3 + \log t$. Blue ones show some connections between nodes representing same-colored edges. Orange ones show some connections between nodes representing incident edges. Black ones illustrate the between-layer connections that force copying the selection made at the second copy nodes of the previous layer. Black dotted edges show the connectivity of special nodes. Reservoir edges for copy 2 in layer 2 connect to all second copy nodes in layer 2 of this instance component and all other instance components. Only one pool of reservoir edges is drawn for clarity but four exist.

Since no token will be able to reach the special nodes of a layer (or the local special nodes of the instance component) within our budget, the special nodes of each layer (and eventually the local special nodes of the instance component for j) will only be disconnected by filling edge nodes of the layer with tokens. For that, a set (which we refer to as a *pool for copy 1 in layer ℓ*) of $m - 1$ reservoir edges for copy 1 in layer ℓ and another pool for

copy 2 in layer ℓ of $m - 1$ *reservoir edges for copy 2 in layer ℓ* are created with tokens on all their nodes. In each reservoir edge, we identify one endpoint of the edge as a *reservoir node*. The $m - 1$ *reservoir nodes for copy 1 in layer ℓ* will be made adjacent to the first copy nodes in layer ℓ of each instance component. The $m - 1$ *reservoir nodes for copy 2 in layer ℓ* will be made adjacent to the second copy nodes in layer ℓ of each instance component.

Additionally, each reservoir node will connect through $m^3 + \log q$ -length paths to all local special s and t nodes in all instance components. This will ensure that these reservoir nodes are included in any vertex cut (since the local special s and t nodes of at least one instance component will always be token-free and reachable from the global special s and t nodes).

The fact that all reservoir nodes must be in T will make it so that only $m - 1$ tokens on the reservoir nodes for copy 1 in layer ℓ can leave their reservoir edges and go to fill $m - 1$ of the first copy nodes in layer ℓ of the selected instance component. Similarly, only $m - 1$ tokens on the reservoir nodes for copy 2 in layer ℓ can leave their reservoir edges and go to fill $m - 1$ of the second copy nodes in layer ℓ of the selected instance component. The other $m - 1$ tokens of any pool will only be used to refill the empty reservoir nodes, using one token slide per token.

Note that the tokens on selector nodes and those in the pools will be the only tokens available, and the budget will be set so that given the construction, only the described token slides can occur. However, this will mean that in each layer of the selected instance component, at most $m - 1$ first copy nodes and at most $m - 1$ second copy nodes can be filled with tokens. This implies that in each layer of the selected instance component, at least two nodes, one being a first copy node of an edge and the other being a second copy node of a certain other edge, remain token-free in the final configuration. We exploit this fact when forming the connections inside an instance component so that each layer encodes the selection of one edge to be part of the rainbow matching (except for the first layer, which encodes the selection of two edges) by leaving one of the nodes representing it token-free.

Connections inside an instance component must ensure three properties: first, that its local special s and t nodes need to be disconnected; second, that the special s and t nodes of each layer within the instance component need to be disconnected; and third, that the token-free non-special nodes remaining in the layers encode a valid rainbow matching. However, these connections must not allow for any pattern of token sliding that is different from the one we described already (where $m - 1$ tokens from each pool slide to adjacent edge nodes). For this reason, each such connection inside an instance component will be a

path of length $m^3 + \log q$ (exceeding our budget, and thus preventing any token traversal).

In an instance component for j , we connect the local special s node for j to the special s nodes for each layer of component j , and the local special t node for j to the special t nodes for each layer of the same component. Additionally, we connect the special s node for a layer of component j to all first copy nodes in the same layer, and the special t node for a layer of component j to all second copy nodes in the same layer. See Figure 5.11. The critical connections that enforce our rainbow matching constraints are as follows.

- For edges $e, e' \in E(G^j)$ of the same color, we connect the first copy (e, i, j) to the second copy (e', i', j) for each $i' \geq i$ (note that this means that first copy (e, i, j) and second copy (e', i', j) cannot remain both token-free in the final configuration; otherwise, we would have a token-free path between the special s node for layer i of component j and the special t node for layer i' of component j . This is exactly what we need so that the edge nodes that remain token-free in the selected instance component encode edges of a rainbow matching as these nodes represent same color edges that must not be selected together.),
- For adjacent edges $e, e' \in E(G^j)$, we also connect the first copy (e, i, j) to the second copy (e', i', j) for each $i' \geq i$ (note that this similarly means that first copy (e, i, j) and second copy (e', i', j) cannot remain both token-free in the final configuration, which is what we need to encode a rainbow matching using the edge nodes that remain token-free in the selected instance component),
- In all but the first layer, the set of first copy nodes will only copy the edge selection that occurred at the set of second copy nodes of the previous layer. To do so, between consecutive layers i and $i + 1$, we connect the second copy (e, i, j) to the first copy $(e', i + 1, j)$ for all $e' \neq e$ (note that this implies if the second copy (e, i, j) remains token-free in the final configuration, then leaving anything but the first copy $(e, i + 1, j)$ in layer $i + 1$ token-free will leave an uncut path between the local special s and t nodes for j).

Recall that we place tokens on all selector nodes for the tree structure (exactly $\log q$ tokens) and on all nodes of the reservoir edges (exactly $4(\kappa - 1)(m - 1)$ tokens). The budget is set precisely to $b = \log q + 4(\kappa - 1)(m - 1)$, which allows each of the tokens one token slide as we have described and is less than the length of our long connecting paths.

The budget and the fact that this construction runs in polynomial time with respect to the total size of all q instances ensures that we obtain a valid or-cross-composition. We now present the forward and backward directions of the proof of Theorem 70.

Claim 71. *If for some $j \in [q]$, $((G^j, \mathcal{C}^j), \kappa^j)$ is a yes-instance of RAINBOW MATCHING, then the constructed instance $(H, S, b, \text{global special } s \text{ node}, \text{global special } t \text{ node})$ is a yes-instance of VERTEX CUT-DISCOVERY.*

Proof of Claim 71. Let $RM = \{e_1, e_2, \dots, e_\kappa\}$ be a solution to the instance $((G^j, \mathcal{C}^j), \kappa^j)$. Thus, RM forms a matching in G^j where all edges have distinct colors. We apply the following token slides to disconnect the global special s node from the global special t node.

First, we use the $\log q$ selector tokens in the binary tree to “select” instance component H^j . Each selector token at depth d slides to block one branch, collectively leaving only the paths from the global special s node to the global special t node that pass through component H^j unblocked. This uses exactly $\log q$ token slides.

Next, we disconnect the local special nodes for j by disconnecting the special nodes of each layer of that component. For the first layer (layer 1), which encodes two edges, we slide the $m - 1$ tokens from the reservoir nodes for copy 1 in layer 1 onto all first copy nodes in layer 1 of component j except the first copy $(e_1, 1, j)$. Also, we slide the $m - 1$ tokens from the reservoir nodes for copy 2 in layer 1 onto all second copy nodes in layer 1 of component j except the second copy $(e_2, 1, j)$. For each subsequent layer $\ell \in \{2, 3, \dots, \kappa - 1\}$, which each encode one edge:

- We slide the $m - 1$ tokens from the reservoir nodes for copy 1 in layer ℓ onto all first copy nodes in layer ℓ of component j except the first copy $(e_{\ell+1}, \ell, j)$.
- We slide the $m - 1$ tokens from the reservoir nodes for copy 2 in layer ℓ onto all second copy nodes in layer ℓ of component j except the second copy $(e_{\ell+1}, \ell, j)$.

This uses $2(m - 1)$ token slides per layer, totaling $2(\kappa - 1)(m - 1)$ token slides.

As each reservoir node is vacated, the token on the other endpoint of its reservoir edge slides onto it, maintaining these nodes in the vertex cut. This is necessary as no tokens will be on the local special s and t nodes for j in T , which means that if a reservoir node does not have a token, there will be a path between the global special s and t nodes that passes through it. This uses $2(\kappa - 1)(m - 1)$ token slides in total.

Following these steps, we have executed exactly $b = \log q + 4(\kappa - 1)(m - 1)$ token slides.

To see that the global special nodes are now disconnected, observe that after the tree selection, all paths must go through the local special nodes for j . Any such path must traverse at least one layer (from a special s node for a layer in component j to a first copy

node in one layer of component j to a second copy node in the same or another layer of component j), or a reservoir node. All reservoir nodes have tokens. Between layers ℓ and $\ell - 1$ the unfilled first copy node in layer ℓ of component j and the unfilled second copy node in layer $\ell - 1$ of component j represent the same edge by our choice of token slides and thus have no $m^3 + \log q$ -length connection between them by our construction. Additionally, by our construction there exists no $m^3 + \log q$ -length connection from a first copy node to a second copy node representing edges that are of different colors, or non-adjacent.

Since we cannot find a path from the local special s node for j to the local special t node for j , the final configuration cuts the local special s and t for j , and thus the global special nodes. $\square$

Claim 72. *If $(H, S, b, \text{global special } s \text{ node}, \text{global special } t \text{ node})$ is a yes-instance of VERTEX CUT-DISCOVERY, then there exists an integer $j \in [q]$, such that $((G^j, \mathcal{C}^j), \kappa^j)$ is a yes-instance of RAINBOW MATCHING.*

Proof of Claim 72. Let $(T, \vec{T})$ be a solution whose final configuration T disconnects the global special nodes using exactly $b = \log q + 4(\kappa - 1)(m - 1)$ token slides. Since all connections use paths of length $m^3 + \log t > b$, we know that each token on a selector node that slides does so at most once. Consider the $\log q$ selector tokens in the binary tree structure. Each selector token at depth d can block at most one of the two branches at its depth. With $\log q$ tokens across all depths, they can collectively block at most $q - 1$ of the q paths from the global special s node through the tree to the instance components. Therefore, at least one instance component must remain unblocked. Let H^j be an instance component whose path from the global special s node remains unblocked by the selector tokens.

After this selection phase, there remain paths from the global special s node to the global special t node that pass through the local special s node and the local special t node for j . Since the distance from any token's initial position to any special node in H^j (local or layer-specific) exceeds the budget b , no token can reach these special nodes directly. Therefore, disconnection must occur by placing tokens on the edge nodes in the layers of the instance component H^j .

Consider the reservoir nodes. Each reservoir node is connected via paths of length $m^3 + \log q$ to all local special nodes in all instance components. Since the local special nodes in at least one instance component H^j remain token-free and reachable from the global special nodes, every reservoir node must be included in any vertex cut T .

For each layer $\ell \in [\kappa - 1]$, there are two pools of $m - 1$ reservoir edges (one for copy 1, one for copy 2). Initially, all $2(m - 1)$ nodes of these reservoir edges contain tokens.

When a reservoir node slides its token into instance component H^j , at least one additional token slide is required to fill this reservoir node and maintain it in the cut. In a solution that requires the minimum number of token slides, we can safely assume that this reservoir node is filled by the token on the other endpoint of its reservoir edge in one token slide, and we do assume as such.

Since the connections inside each instance component are of length $m^3 + \log t$, which is greater than the budget, each token sliding from a reservoir node of a pool to an adjacent edge node in H^j will have to end up in T on one of the edge nodes adjacent to the reservoir nodes of the same pool (notice that using even one token slide to slide a token into an $m^3 + \log q$ -length connection path is redundant as we cannot traverse this path and thus, from the perspective of creating a cut, it is enough to remain at the edge node). Given that we have only $m - 1$ reservoir edges per pool, we can only have $m - 1$ tokens on $m - 1$ first copy nodes and $m - 1$ tokens on $m - 1$ second copy nodes in each layer of component j in T . Therefore, since we have m first copy nodes and m second copy nodes in each layer of component j , at least two nodes must remain unfilled in each layer of component j in T : one first copy node and one second copy node.

We claim that the set of edges represented by the unfilled first and second copy nodes in layer 1 of component j and the unfilled second copy nodes in all other layers of component j forms a rainbow matching. Suppose for contradiction that these unfilled nodes violate the rainbow matching constraints. First, consider the case where the unfilled first copy of edge e in layer 1 of component j and the unfilled second copy of edge e' in layer $\ell \geq 1$ of component j are such that $e = e'$, or e is adjacent to e' , or e and e' have the same color. Then by construction, there exists an uncut path from the global special s node to the global special t node that passes through these two nodes and their $m^3 + \log q$ -length connection path, contradicting that T is a cut.

Second, consider the case where the unfilled second copy of edge e in layer ℓ of component j and the unfilled second copy of edge e' in layer $\ell' > \ell$ of component j are such that $e = e'$, or e is adjacent to e' , or e and e' have the same color. Then, the unfilled first copy in layer $\ell + 1$ of component j must represent edge e . Otherwise, by construction, there would exist an uncut path through the local special s node for j , the special s node for layer $\ell + 1$ in component j , the unfilled first copy in layer $\ell + 1$ of component j , the unfilled second copy of edge e in layer ℓ of component j , the special t node for layer ℓ in component j , and finally the local special t node for j , contradicting that T is a cut. However, this case would also imply, by construction, that there exists an uncut path from the global special s node to the global special t node that goes through the unfilled first copy of edge e in layer $\ell + 1$ and the unfilled second copy of edge e' in layer ℓ' (since $\ell' > \ell$), again contradicting that T is a cut.

Thus, we have κ unfilled nodes representing pairwise non-adjacent edges of different colors, yielding a rainbow matching of size κ in G^j . $\square$

5.3 Shortest Path-Discovery

In this section, we shift our focus to SHORTEST PATH-DISCOVERY, completing with this our study of solution discovery variants of fundamental polynomial-time solvable problems. We show in Section 5.3.1 that the problem is NP-hard on a sparse graph class through a reduction from HAMILTONIAN PATH (which we define in the same section). In Section 5.3.2, we show that the problem does not admit a polynomial kernel with respect to $k + b$, unless $\text{NP} \subseteq \text{coNP}/\text{P}$, through an or-cross-composition that handles multiple HAMILTONIAN PATH instances and builds upon the reduction of the preceding section. In Section 5.3.3, we present a short proof that the problem is fixed-parameter tractable when parameterized by k by reducing it to finding shortest paths in weighted directed acyclic graphs, where edge weights encode token sliding costs. This happens after the algorithm considers all $k!$ orderings of tokens to vertices at increasing distances from the source, exploiting that any discoverable shortest path contains exactly k vertices. Finally, in Section 5.3.4, we show that the problem is also fixed-parameter tractable using similarly a distance-based structure and noticing that in a yes-instance and assuming b is much smaller than k , most tokens must already be at distinct distances from the input vertices, forming path sub-segments, and need not slide.

5.3.1 NP-hardness of Shortest Path-Discovery

In this section, we show that SHORTEST PATH-DISCOVERY is NP-hard even on 2-degenerate bipartite graphs. In particular, we show the following.

Theorem 73. *SHORTEST PATH-DISCOVERY is NP-hard on 2-degenerate bipartite graphs.*

To prove Theorem 73, we reduce from the HAMILTONIAN PATH problem. In the HAMILTONIAN PATH problem, we are given a graph G and the goal is to decide whether G contains a Hamiltonian path, which is a path that visits every vertex exactly once. The problem is NP-complete [93].

Let G be an instance of HAMILTONIAN PATH with vertices $V(G) = \{v_1, \dots, v_n\}$. For clarity, we refer to *vertices* in the instance of HAMILTONIAN PATH and *nodes* in our construction.

We construct a graph H with special nodes s and t between which we seek a shortest path. For each $v \in V(G)$, we add a *starting path for v* consisting of three nodes, where the first node is adjacent to s . Similarly, for each $v \in V(G)$, we add an *ending path for v* consisting of three nodes, where the third node is adjacent to t . These paths ensure that one vertex is selected to be the starting vertex and one vertex is selected to be the ending vertex of the Hamiltonian path.

We create a *starting edge verifier path* consisting of three nodes, where the middle node is adjacent to the middle nodes of all starting paths. We create an *ending edge verifier path* consisting of three nodes, where the middle node is adjacent to the middle nodes of all ending paths. See Figure 5.12.

For each $i \in [n]$, we create a *layer i* consisting of n *select nodes*. In each layer, each select node corresponds to one vertex in G . The select node of layer i with a token in the solution will indicate the i th vertex of the Hamiltonian path. For each $v \in V(G)$, we make the third node of the starting edge verifier path for v adjacent to the select node corresponding to v in layer 1 and the first node of the ending edge verifier path for v adjacent to the select node corresponding to v in layer n .

For each $i < n$ and each edge $uv \in E(G)$, we create a *transition path* consisting of three nodes connecting the select node corresponding to u in layer i to the select node corresponding to v in layer $i+1$. Similarly, we create another transition path of three nodes connecting the select node corresponding to v in layer i to the select node corresponding to u in layer $i+1$.

For each $i < n$, we create an *edge verifier path for layer i* consisting of three nodes with tokens, where the middle node is adjacent to the middle node of all transition paths between layer i and layer $i+1$.

For each $v \in V(G)$, we create an *index decider node for v* (corresponding to v) connected to all select nodes corresponding to vertex v via paths of $2n+1$ internal nodes.

Initially, we place tokens on the n index decider nodes, all nodes in all edge verifier paths (three nodes per path for $n+1$ paths), s , and t , and set the budget to $b = n(2n+2)+7(n+1)$.

The key insight is that the tokens on the index decider nodes select the position of their corresponding vertices in the Hamiltonian path (an index decider node for v selects position i by sliding its token to the select node for v in layer i , thus requiring $2n+2$ token slides). Tokens on each edge verifier path verify that a valid transition path exists between two consecutive layers by sliding their tokens to the transition path.

It is easy to see that the reduction takes polynomial time with respect to $|V(G)|$.

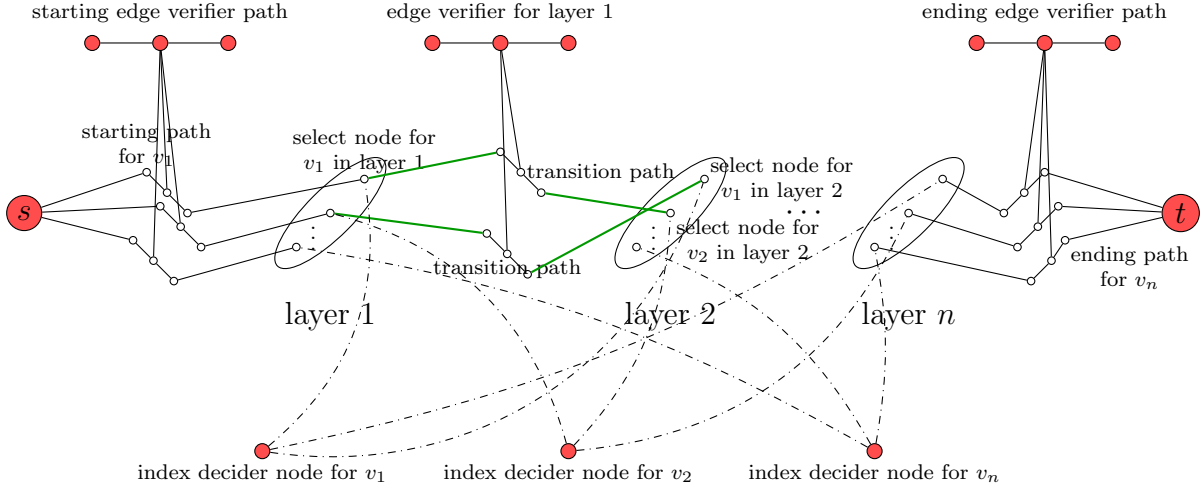


Figure 5.12: The reduced graph H for a HAMILTONIAN PATH instance on graph G with n vertices. Green edges show how transition paths connect: if G contains edge v_1v_2 , then a transition path of three nodes connects to the select node for v_1 in layer i and the select node for v_2 in layer $i+1$, and another transition path of three nodes connects to the select node for v_2 in layer i and the select node for v_1 in layer $i+1$. Starting and ending paths also consist of three nodes each. Edge verifier paths have three nodes with tokens, where the middle node is adjacent to the middle nodes of all relevant paths. Red nodes initially have tokens. Dashed lines represent paths with $2n+1$ internal nodes.

Lemma 74. H is a 2-degenerate bipartite graph.

Proof of Lemma 74. First, we show that H is bipartite. Recall that a graph is bipartite if we can partition its vertices into two partite sets such that no two vertices that belong to the same set are adjacent. We construct the bipartition (L, R) of $V(H)$ as follows. We place in L nodes s, t , all select nodes, all index decider nodes, the middle node of each transition, starting, and ending path and the endpoints of each edge verifier path. We place in R the middle node of each edge verifier path, and the first and third node of each transition, starting, and ending path. For each path connecting an index decider node to a select node, we place internal nodes alternately in L and R starting by placing the node adjacent to the select node in R . Since all edges go between L and R , the graph H is bipartite.

To show that H is 2-degenerate, we prove that every non-empty subgraph of H contains a vertex of degree at most two. Consider any non-empty subgraph $H' \subseteq H$. If H' contains any non-middle node from a transition path, starting path, or ending path, such nodes

have degree exactly two in H , thus also in H' . If H' contains any internal node of a path that connects an index decider node to a select node, such nodes have degree exactly two in H , thus also in H' . Otherwise, H' is a forest and contains at least one vertex of degree at most one. Therefore, every non-empty subgraph of H contains a vertex of degree at most two, proving that H is 2-degenerate. $\square$

Lemma 75. *If G has a Hamiltonian path, then (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY.*

Proof of Lemma 75. Let $v_{i_1}, v_{i_2}, \dots, v_{i_n}$ be a Hamiltonian path in G . We show how to move tokens to form a shortest path from s to t .

We move the tokens from the first, middle and third nodes of the starting edge verifier path to the first, middle, and third nodes, respectively, of the starting path for v_{i_1} . We move the tokens from the first, middle and third nodes of the ending edge verifier path to the first, middle, and third nodes, respectively, of the ending path for v_{i_n} . This costs 14 token slides.

For each $j \in [n]$, we move the token from the index decider node for v_{i_j} to the select node corresponding to v_{i_j} in layer j . This costs $n(2n + 2)$ token slides.

For each $j < n$, since $v_{i_j}v_{i_{j+1}}$ is an edge in G , there exists a transition path connecting the select node corresponding to v_{i_j} in layer j to the select node corresponding to $v_{i_{j+1}}$ in layer $j + 1$. We move the tokens from the first, middle and third nodes of the edge verifier path for layer j to the first, middle and third nodes of this transition path. This costs $7(n - 1)$ token slides.

Any shortest path must pass through one starting path (three nodes), exactly one select node from each layer connected through transition paths (each of three nodes), and one ending path (three nodes). Thus, the path formed of the vertices with tokens is a shortest path between s and t and the number of token slides needed is $14 + n(2n + 2) + 7(n - 1) = b$ and (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY. $\square$

Lemma 76. *If (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY, then G has a Hamiltonian path.*

Proof of Lemma 76. Let T be a shortest path between s and t discoverable from S with a budget b . Any shortest path must pass through one starting path (three nodes), exactly one select node from each layer connected through transition paths (each of three nodes), and one ending path (three nodes). The number of vertices in a minimum path length is therefore $1 + 3 + n + 3(n - 1) + 3 + 1 = 4n + 5$.

Each of the n tokens on the index decider nodes moves at least to a select node (cost: $n(2n + 2)$). Each edge verifier path must move its three tokens at least to the middle node of a closest transition, starting, or ending path (cost: $5(n + 1)$). The first and third nodes of each such starting, transition, and ending path remain token-free, while the middle node contains two extra tokens. Thus, with a remaining budget of only $2(n + 1)$ token slides, the first and third nodes of each starting, transition, and ending path are filled using one token slide each, with the tokens on the middle node of the same path.

Given the construction, the shortest path induced by the vertices in T must be formed of a sequence of vertices $v_{i_1}, v_{i_2}, \dots, v_{i_n}$ corresponding to the select nodes with tokens in layers 1 through n , where each two consecutive vertices share an edge in G . Additionally, given that each index decider node can slide its token to a select node representing a distinct vertex in G , this sequence forms a Hamiltonian path in G . $\square$

5.3.2 No Polynomial Kernel for Shortest Path-Discovery with Parameter $k + b$

In this section, we show that SHORTEST PATH-DISCOVERY when parameterized by $k + b$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/P$. We do so by providing an or-cross-composition from the HAMILTONIAN PATH problem. Thus, we prove the following theorem.

Theorem 77. *There exists an or-cross-composition from HAMILTONIAN PATH to SHORTEST PATH-DISCOVERY parameterized by $k+b$. Consequently, SHORTEST PATH-DISCOVERY does not admit a polynomial kernel with respect to $k + b$, unless $\text{NP} \subseteq \text{coNP}/P$.*

The designed or-cross-composition builds upon the reduction given for the NP-hardness proof of SHORTEST PATH-DISCOVERY to handle multiple instances simultaneously while ensuring that exactly one instance can be solved.

Let $\mathcal{R}$ be the polynomial equivalence relation where two graphs G and H are equivalent if and only if $|V(G)| = |V(H)|$. This relation can be decided in polynomial time and partitions HAMILTONIAN PATH instances into classes based on their numbers of vertices.

Let $G^1, \dots, G^q$ be instances of HAMILTONIAN PATH that belong to the same equivalence class, where each G^j has vertex set $V(G^j) = \{v_1^j, \dots, v_n^j\}$. We construct an instance (H, s, t, S, b) of SHORTEST PATH-DISCOVERY such that it is a yes-instance if and only if at least one G^j admits a Hamiltonian path. We adapt the construction from Theorem 73 to accommodate multiple instances while sharing the critical resources.

In H , we let the following be global components shared among all instances: the special nodes s and t , the *index decider node* for v_i for each $i \in [n]$, the *starting edge verifier path* consisting of three nodes with tokens, the *ending edge verifier path* consisting of three nodes with tokens, and the *edge verifier path for layer i* for each $i \in [n - 1]$ consisting of three nodes with tokens. See Figure 5.13. Here, v_i denotes the i th vertex on the Hamiltonian path of the solution.

For each instance G^j , for $j \in [q]$, we will have the following: for each $i \in [n]$, a *layer* (j, i) consisting of n *select nodes*, each one corresponding to a distinct vertex in $V(G^j)$. We denote the select node in layer (j, i) corresponding to vertex v_k^j for some $k \in [n]$ as the *select node* (j, i, k) . Then, for each $v_k^j \in V(G^j)$, we will have a *starting path for (j, k)* consisting of three nodes, where the first node is adjacent to s and the third node is adjacent to the select node $(j, 1, k)$. Similarly, for each $v_k^j \in V(G^j)$, we will have an *ending path for (j, k)* consisting of three nodes, where the first node is adjacent to the select node (j, n, k) and the third node is adjacent to t . Finally, for each $i \in [n - 1]$ and each edge $v_k^j v_{k'}^j \in E(G^j)$, we will have two *transition paths*: one consisting of three nodes from select node (j, i, k) to $(j, i + 1, k')$, and another from (j, i, k') to $(j, i + 1, k)$.

Lastly, we add the following connections to H . We make the middle node of the starting edge verifier path adjacent to the middle node of every starting path across all instances. Similarly, we make the middle node of the ending edge verifier path adjacent to the middle node of every ending path across all instances. For each $i \in [n - 1]$, we make the middle node of the edge verifier path for layer i adjacent to the middle node of every transition path between layer (j, i) and layer $(j, i + 1)$ for all $j \in [q]$. In the end, for each $v_i \in \{v_1, \dots, v_n\}$ and each instance $j \in [q]$, we make the index decider node for v_i connected to all select nodes (j, ℓ, i) for $\ell \in [n]$ via paths of the internal nodes $2n + 1$.

As in the reduction, the tokens are initially placed on s , t , all n index decider nodes, and all $3(n + 1)$ nodes in the edge verifier paths. Thus, $|S| = 2 + n + 3(n + 1) = 4n + 5$. We set the budget to $b = n(2n + 2) + 7(n + 1) = 2n^2 + 9n + 7$.

Note that $k + b = 2n^2 + 13n + 12$, thus the parameter is bounded above by a polynomial in the size of the maximum size instance. The size of H is $O(qn^4)$: we have q instances, each contributing $n^2 + 6n + 3m(n - 1)$ nodes for layers (n layers of n nodes each) and paths ($2n$ starting and ending paths of three nodes each and $m(n - 1)$ transition paths of three nodes each), plus $n^2(2n + 1)$ nodes for connections from index decider nodes (each of the n index decider nodes connects to n select nodes via paths of $2n + 1$ nodes). Since each input instance has a size of at least n , the total size of H and the running time of the or-cross-composition is polynomial in $\sum_{j=1}^q |G^j|$.

Moreover, note that H remains **2-degenerate** and **bipartite**.

Lemma 78. *H is 2-degenerate and bipartite.*

Proof of Lemma 78. We construct a bipartition (L, R) of $V(H)$ as follows. We place the following in L : nodes s and t , all select nodes, all index decider nodes, the middle node of each transition/starting/ending path, and the endpoints of each edge verifier path. We place the following in R : the middle node of each edge verifier path, and the first and third nodes of each transition/starting/ending path. For paths connecting index deciders to select nodes, we place internal nodes alternately in L and R , starting with the node adjacent to the select node in R . Since all edges connect nodes between L and R , the graph H is bipartite.

To prove H is 2-degenerate, we show that every non-empty subgraph $H' \subseteq H$ contains a vertex of degree at most two. Consider any non-empty subgraph H' . If H' contains any non-middle node from a transition, starting, or ending path, such nodes have degree exactly two in H (and thus at most two in H'). If H' contains any internal node from a path connecting an index decider node to a select node, such nodes also have degree exactly two. Otherwise, H' contains only nodes that induce a forest, thus H' must contain a node of degree at most one. Therefore, H is 2-degenerate. $\square$

We now prove that (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY if and only if at least one G^j has a Hamiltonian path.

Lemma 79. *If G^j for some $j \in [q]$ has a Hamiltonian path, then (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY.*

Proof of Lemma 79. Suppose G^j has a Hamiltonian path $v_{i_1}^j, v_{i_2}^j, \dots, v_{i_n}^j$. We show how to discover a shortest s - t path using exactly the budget b .

First, we move the tokens from the starting edge verifier path to the starting path for $(j, v_{i_1}^j)$: the three tokens slide to the corresponding three nodes of this path. Similarly, we move the tokens from the ending edge verifier path to the ending path for $(j, v_{i_n}^j)$. This requires $7 \cdot 2 = 14$ token slides.

For each $\ell \in [n]$, we move the token from the index decider node for v_{i_ℓ} to the select node (j, ℓ, i_ℓ) in layer (j, ℓ) . Since each such token traverses a path of $2n + 1$ internal nodes, this requires $n(2n + 2)$ token slides in total.

For each $\ell \in [n - 1]$, since $v_{i_\ell}^j v_{i_{\ell+1}}^j \in E(G^j)$, there exists a transition path connecting select node (j, ℓ, i_ℓ) to select node $(j, \ell + 1, i_{\ell+1})$. We move the three tokens from the edge verifier path for layer ℓ to the three nodes of this transition path. This requires $7(n - 1)$ token slides.

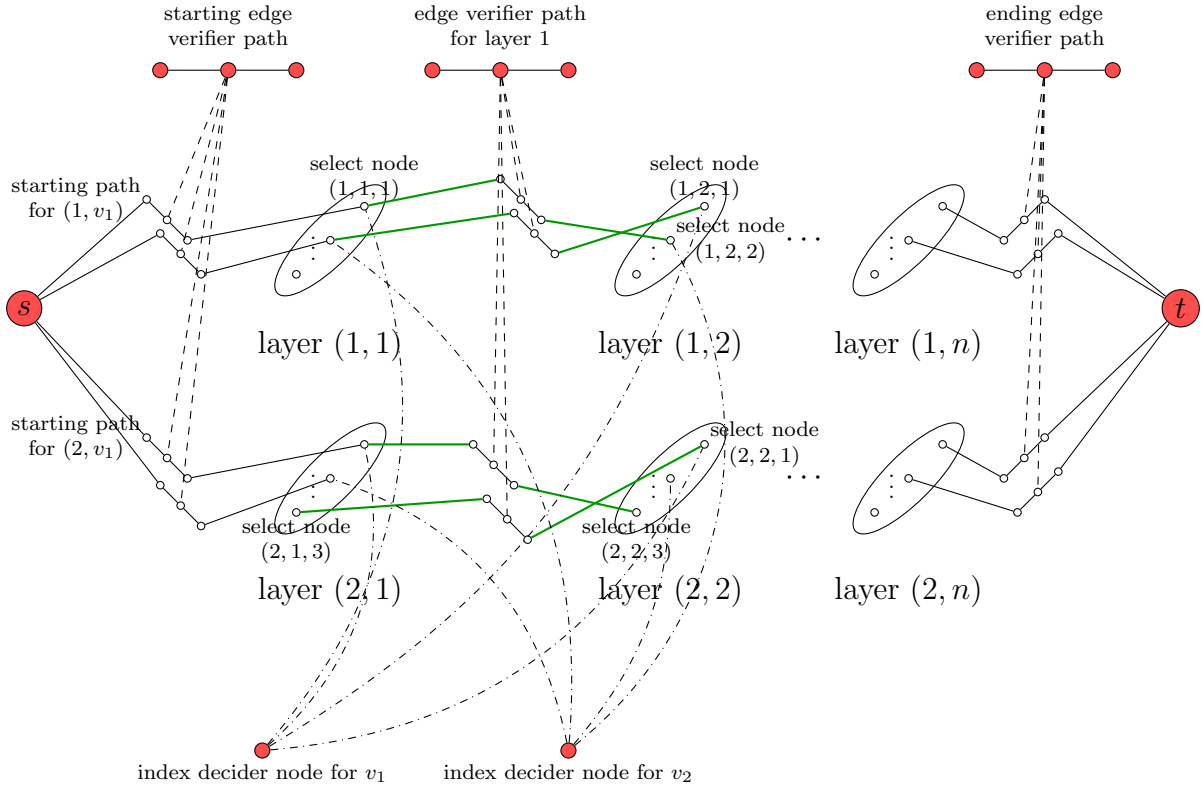


Figure 5.13: Or-cross-composition of two instances G_1 and G_2 . Red nodes have initial tokens. Green edges show how transition paths connect select nodes between consecutive layers: if G_1 contains edge v_1v_2 , then a transition path connects the select node $(1, 1, 1)$ to the select node $(1, 2, 2)$ and another transition path connects the select node $(1, 1, 2)$ to the select node $(1, 2, 1)$. Dashed lines represent edges that connect edge verifier paths to all relevant paths across both instances. Dash-dotted lines represent paths with $2n + 1$ internal nodes.

The discovered path traverses s , the starting path for $(j, v_{i_1}^j)$ (three nodes), the select nodes $(j, 1, i_1), (j, 2, i_2), \dots, (j, n, i_n)$ connected by transition paths (each of three nodes), the ending path for $(j, v_{i_n}^j)$ (three nodes), and finally t . This gives a path of $1 + 3 + n + 3(n - 1) + 3 + 1 = 4n + 5$ nodes, which is indeed a shortest s - t path in H . The total number of token slides is $14 + n(2n + 2) + 7(n - 1) = 2n^2 + 9n + 7 = b$. $\square$

Lemma 80. *If (H, s, t, S, b) is a yes-instance of SHORTEST PATH-DISCOVERY, then there exists $j \in [q]$ such that G^j has a Hamiltonian path.*

Proof of Lemma 80. Suppose (H, s, t, S, b) is a yes-instance. We show that some G^j must have a Hamiltonian path.

Any shortest s - t path must traverse exactly one starting path, one select node from each of n layers (all from the same instance) connected by transition paths, and one ending path, giving a path of a total of $1 + 3 + n + 3(n - 1) + 3 + 1 = 4n + 5$ nodes.

Each of the n index decider nodes must move its token to a select node (cost: $n(2n+2)$). Each edge verifier path must move its three tokens to the middle node of a closest transition, starting, or ending path (cost: $5(n + 1)$).

Crucially, the first and third nodes of each such starting, transition, or ending path remain token-free, while the middle node contains two extra tokens. Thus, with a remaining budget of only $2(n + 1)$ token slides, the first and third nodes of each starting, transition, and ending path are filled using one token slide each, with the tokens on the middle node of the same path.

Given the construction, the discovered shortest path determines a sequence of vertices $v_{i_1}^j, v_{i_2}^j, \dots, v_{i_n}^j$ belonging to the same instance G^j and corresponding to the select nodes with tokens in layers $(j, 1)$ through (j, n) , where consecutive vertices $v_{i_\ell}^j$ and $v_{i_{\ell+1}}^j$ share an edge in G^j (otherwise no transition path would exist between the corresponding select nodes). Additionally, since each index decider node slides its token to a select node representing a distinct vertex in $V(G^j)$, all vertices in the sequence are distinct. Thus, $v_{i_1}^j, v_{i_2}^j, \dots, v_{i_n}^j$ forms a Hamiltonian path in G^j . $\square$

5.3.3 Fixed-Parameter Tractability of Shortest Path-Discovery for Parameter k

In this section, we prove the following theorem.

Theorem 81. *SHORTEST PATH-DISCOVERY is fixed-parameter tractable with respect to parameter k .*

To prove Theorem 81, we present an algorithm that reduces SHORTEST PATH-DISCOVERY to finding a shortest path in a weighted acyclic digraph.

Proof of Theorem 81. Let (G, s, t, S, b) be an instance of SHORTEST PATH-DISCOVERY. The algorithm proceeds as follows.

We first calculate the distance from s to every vertex $v \in V(G)$ (using [breadth-first search](#)). Since any discoverable shortest path between s and t has at most k vertices, we delete each vertex whose distance from s exceeds $k - 1$.

We consider the $k!$ ways to order the vertices in S such that if $s_0 s_1 \dots s_{k-1}$ is the order, then the token on s_i will be moved to a vertex at distance exactly i from s (from the problem definition, we have that the token on s_0 will be moved to s and the token on s_{k-1} will be moved to t).

For each such ordering, we construct a weighted acyclic digraph H as follows. For each edge $uv \in E(G)$ where u is at distance i from s and v is at distance $i + 1$, we orient the edge as (u, v) , place it in H , and assign the distance between s_{i+1} and v (which represents the number of token slides needed to move the token at the vertex s_{i+1} to the vertex v) as its weight.

Since H only contains arcs from distance i vertices to distance $i + 1$ vertices for each $i \in \{0, \dots, k-2\}$, it forms an acyclic digraph where every path from s to t in H corresponds to a shortest path between s and t in the original graph G . Moreover, the weight of a path in H equals the total cost of moving tokens to the vertices of the path, minus the initial cost of sliding the token on s_0 to s .

We compute a shortest weighted path P from s to t in H using [Dijkstra's algorithm](#) in $\mathcal{O}((n + m) \log n)$ time. The instance (G, s, t, S, b) has a solution if and only if the total weight of P plus the initial cost of sliding the token on s_0 to s is at most b . The algorithm initially does a breadth-first search which has polynomial running time. Then, it considers $k!$ orderings. For each ordering, it constructs a graph H and finds a shortest weighted path in polynomial time as well.

The correctness follows from two key observations: first, any discoverable shortest path from s to t has at most k vertices since we have only k tokens to place on the path, and second, the construction of H ensures that every path from s to t in H corresponds to a shortest path in G while the edge weights correctly capture the minimum token movement costs. By considering all $k!$ possible orderings of tokens, we guarantee that we find the optimal assignment of tokens to vertices that minimizes the total token sliding cost. Therefore, SHORTEST PATH-DISCOVERY is fixed-parameter tractable when parameterized by k , with a total running time of $\mathcal{O}(k(n + m) + k!(n + m) \log n)$, where $\mathcal{O}(k(n + m))$ accounts for the breadth-first searches used to compute distances from each token. $\square$

5.3.4 Fixed-Parameter Tractability of Shortest Path-Discovery for Parameter b

This section is dedicated to the proof of the following theorem.

Theorem 82. *SHORTEST PATH-DISCOVERY is fixed-parameter tractable when parameterized by b .*

Proof. Let (G, s, t, S, b) be an instance of SHORTEST PATH-DISCOVERY. Using [breadth-first search](#), we partition the vertices of G into levels based on their distance from s , where level i contains all vertices at distance exactly i from s . The key intuition behind the algorithm is that if a shortest path can be discovered within a budget of b , then, if b is much smaller than k , most tokens must already be in or near their correct positions.

We say that a level is *very bad* if it contains no tokens or more than one token. Since sliding tokens to or out of very bad levels cannot cost more than the budget, there can be at most b very bad levels, each containing at most $b + 1$ tokens.

For levels containing exactly one token each, we define a level i with token on vertex v to be *bad* if either:

- level $i - 1$ exists, contains exactly one token on vertex u , and $uv \notin E(G)$; or
- level $i + 1$ exists, contains exactly one token on vertex w , and $vw \notin E(G)$.

Since correcting three consecutive bad levels requires at least one unit of budget, we can have at most $3b$ bad levels.

Levels that are neither bad nor very bad are called *good*. Additionally, each bad or very bad level can contain at most $b + 1$ tokens and at most b tokens can initially not belong to a level between s and t . The vertices with tokens in consecutive good levels form paths in G , which we call *good paths*. Note that we have at most $4b + 1$ such good paths as we have at most $4b$ bad or very bad levels that act as separators between good levels.

For any such path $v_1 v_2 \dots v_\ell$ with $\ell \geq 2b + 1$, consider a vertex v_x where $b + 1 \leq x \leq \ell - (b + 1)$. We claim that in any solution, the token on v_x does not move. If it must move, then either all tokens on vertices $v_1, \dots, v_{x-1}$ or all tokens on vertices $v_{x+1}, \dots, v_\ell$ must also move to maintain path connectivity. Since there are at least b tokens in either direction, this would exceed the budget. Therefore, tokens on nodes that are more than b positions away from the endpoints of good paths are *fixed* and never move.

Therefore, the tokens that can potentially move are at most $b(b+1)$ tokens from very bad levels, at most $3b(b+1)$ tokens from bad levels, at most b tokens not on any level between s and t , and at most $2b$ tokens from each boundary of the at most $4b+1$ good paths. This gives at most $4b^2 + 5b + 2b(4b+1) = 12b^2 + 7b$ tokens that can potentially move.

We proceed by guessing which subset of these potentially movable tokens actually move and to which level each moving token is assigned. For each such guess, we construct a weighted acyclic digraph as in the proof of Theorem 81 and in polynomial time compute the minimum number of token slides needed to discover a shortest path. Since we consider at most $\sum_{i=0}^b \binom{12b^2+7b}{i}$ subsets (choosing i tokens for each $i \in \{0, 1, \dots, b\}$) and for each subset of size i , we have $(b+1)^i$ level assignments, the algorithm runs in time $O(f(b) \cdot n^{\mathcal{O}(1)})$ for some computable function f depending only on b . Since we consider at most $\sum_{i=0}^b \binom{12b^2+7b}{i}$ subsets (choosing i tokens for each $i \in \{0, 1, \dots, b\}$) and for each subset of size i , we have $(b+1)^i$ level assignments, the algorithm runs in time $\mathcal{O}((12b^2+7b)^b \cdot (b+1)^b \cdot (n+m) \log n)$ which is $\mathcal{O}(2^{O(b \log b)} \cdot (n+m) \log n)$ and therefore fixed-parameter tractable with respect to b . $\square$

Chapter 6

Solution Discovery for NP-hard Problems

In this chapter, we focus on the solution discovery variants of three fundamental NP-hard problems that have shaped the landscape of theoretical computer science: INDEPENDENT SET, VERTEX COVER, and DOMINATING SET.

These classical problems are particularly compelling for several reasons. First, they represent different facets of graph structure, that is, independence, coverage, and domination. With that, they each capture a real-world phenomenon from conflict-free resource allocation to sensor placement and influence propagation. Second, Fellows et al. [83] established that while these problems exhibit varying behaviors under standard parameterizations (for parameters k and b), they left a crucial question unanswered: what happens when we parameterize by structural parameters?

Previous work showed only that these problems lie in XP when parameterized by treewidth, leaving a significant gap in our understanding. This chapter bridges some of the gap by establishing that all three problems (INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY) are XNLP-hard when parameterized by pathwidth, even when given a path decomposition realizing the pathwidth. These hardness results immediately extend to treewidth, as pathwidth is an upper-bound on treewidth for any graph and they imply that no fixed-parameter tractable algorithm exists for any of these three problems with respect to any of these parameters. The choice to prove hardness for pathwidth is strategic: it yields to a stronger result, as any problem hard for the more restrictive parameter (pathwidth) is necessarily hard for the less restrictive one (treewidth). Our approach employs a unified reduction framework from the

MINIMUM MAXIMUM OUTDEGREE problem, which we define in Section 6.1.

Beyond settling the parameterized complexity with respect to pathwidth and treewidth, we investigate the kernelization complexity of these problems. In Section 6.2, we prove that none of these problems admits a polynomial kernel when parameterized by $b + \text{pathwidth}$, unless $\text{NP} \subseteq \text{coNP}/\text{P}$. This parameter choice is motivated by the goal of identifying the minimal form of structural information that, when combined with the budget parameter b , could allow polynomial kernelization where fixed-parameter tractability exists. Finally, while Fellows et al. showed that INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY admit fixed-parameter tractable algorithms when parameterized by k on specific sparse graph classes (nowhere dense and biclique-free, respectively), we provide polynomial kernels for the same in Sections 6.3 and 6.4.

The results presented in this chapter appeared in the paper titled “*Kernelization Complexity of Solution Discovery Problems*” [104] and co-authored with Mario Grobler, Amer E. Mouawad, Naomi Nishimura, Vijayaragunathan Ramamoorthi, and Sebastian Siebertz [52].

6.1 XNLP-hardness of Independent Set-Discovery, Vertex Cover-Discovery, and Dominating Set-Discovery for Parameter pathwidth

In this section, we show that all of INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY are XNLP-hard when parameterized by pathwidth , even when given a path decomposition that realizes the pathwidth.

6.1.1 The reduction framework

Our reductions are from the MINIMUM MAXIMUM OUTDEGREE (MMO) problem. We begin by noting that an *orientation* of an undirected graph H is an assignment of directions to its edges, transforming each undirected edge $\{u, v\}$ into a directed edge (u, v) or (v, u) . In the MINIMUM MAXIMUM OUTDEGREE (MMO) problem, we are given a weighted graph G , an edge weight function $\omega : E(G) \rightarrow \mathbb{N}$, and a positive integer r (r and the weights are encoded in unary). The goal is to determine whether there exists an orientation of G such that for each vertex $v \in V(G)$, the total weight of arcs directed away from v is at most r . Bodlaender et al. [29] proved that MMO is XNLP-hard when parameterized by pathwidth

even when given a path decomposition realizing the pathwidth. Thus, from here on, we assume we are given as part of the input to MMO a path decomposition $\mathcal{P}_G$ of G of width pw .

For an instance $(G, \mathcal{P}_G, \omega, r)$ of MMO, we denote by Σ the total weight of all edges, n the number of vertices, and m the number of edges. Our reductions will construct a graph H that encodes the orientation problem as a token movement problem, where the movement of tokens from their initial positions to a valid final configuration corresponds to choosing an orientation of G .

Core gadget construction

The constructed graph G consists of two main types of gadgets that encode the constraints of the MMO instance. For each edge $e \in E(G)$, we build an *edge-gadget* for e containing a *constraint gadget* that enforces the requirement that e is oriented away from exactly one of its endpoints. For each vertex $v \in V(G)$, we build a *vertex-gadget* that controls how many edges v orients away from itself.

Edge-gadgets. For each edge $e = uv \in E(H)$, the *edge-gadget* for e is made of one *constraint gadget* and $\omega(e)$ *reservoir paths* that will each supply one token to vertex-gadgets of incident vertices which will (for VERTEX COVER-DISCOVERY and DOMINATING SET-DISCOVERY with the help of the tightness of the budget) have a limited capacity to receive tokens.

We denote the i th of the $\omega(e)$ reservoir paths by the *i th reservoir path for e* , and identify one endpoint of the path as the *i th reservoir node for e* . The *constraint gadget for e* also consists of a path, called the *constraint path for e* , with one endpoint adjacent to two adjacent *facilitator nodes*: one *facilitator node* (e, u) and one *facilitator node* (e, v) , whose roles are to facilitate that tokens on the reservoir nodes for e are supplied from the edge-gadget for e to the vertex-gadgets for u and v .

The connections between an edge-gadget for an edge $e \in E(G)$ and the vertex-gadgets for vertices incident to e will ensure tokens flow from the edge-gadget to only one of the vertex-gadgets corresponding to one endpoint of e . This selection of one vertex per edge will encode the orientation in G . Figure 6.1 shows a sketch of how the gadget appears in all reductions.

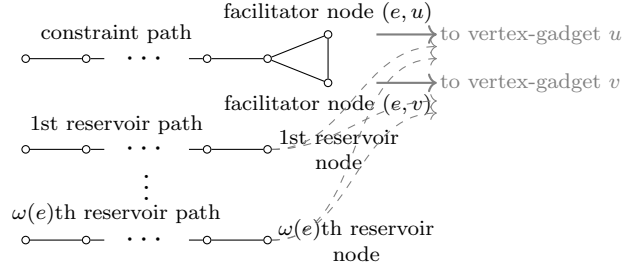


Figure 6.1: An edge-gadget for edge $e = uv$ with weight $\omega(e)$. The constraint gadget ensures that the edge is oriented away from exactly one of its endpoints, while the $\omega(e)$ reservoir paths each provide one token to be distributed. The facilitator nodes (e, u) and (e, v) control token flow to the vertex-gadgets of u and v , respectively. Gray arrows show connections to both vertex-gadgets.

Vertex-gadgets. For each vertex $v \in V(G)$, the *vertex-gadget for v* encodes through the final configuration of tokens how the edges incident to v are oriented in G . The gadget consists of a *central node for v* , which is the endpoint of one *control path for v* (since it will have a fixed token configuration throughout the transformation sequence), and adjacent to r *receiving paths for v* that will each receive at most one token, thus encoding that the vertex v can orient at most r units of edge weight away from itself.

For each edge $e \in E(G)$ incident to v , the vertex-gadget contains $\omega(e)$ *relay paths from v to e* . We denote the i th of the $\omega(e)$ relay paths from v to e by the *i th relay path from v to e* . The role of the i th relay path from v to e is to receive one token from the i th reservoir path for e through its *first checkpoint* (one endpoint of the relay path) and send another of its tokens through the *second checkpoint* (the other endpoint of the relay path) to the central node for v and eventually to one of the r receiving paths. The second checkpoint of each of the relay paths in the gadget is adjacent to the central node of the gadget. We call the node on the i th relay path from v to e adjacent to checkpoint 1 of the path, the *i th guard (v, e)* . See Figure 6.2.

In the final configuration, there are at most r relay paths in the vertex-gadget for a vertex $v \in V(G)$ with a final configuration that is different from the starting configuration and these relay paths are exactly those from edges which we orient away from v in G .

The edge- and vertex-gadgets are connected in a way such that for each edge $e = uv \in E(G)$, the i th relay path from u to e and the i th relay path from v to e can receive a token from the i th reservoir path for e through their first checkpoints, and the guards of the relay paths from v (or u , respectively) to e can communicate (whether they have a token

or not) to the facilitator node (e, v) ((e, u) , respectively) which in turn can “instruct” them to move their tokens out of the corresponding checkpoint 2 nodes.

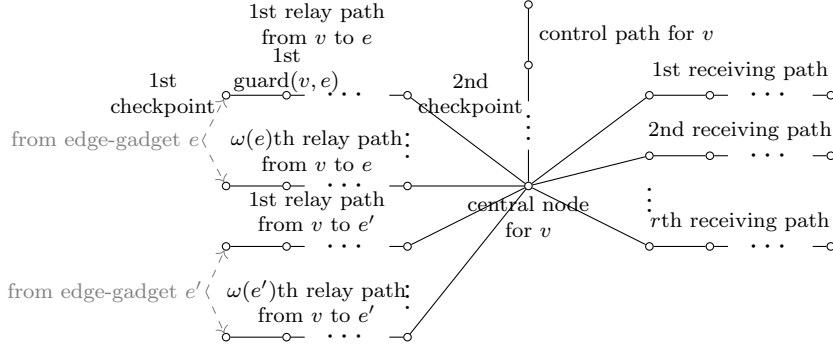


Figure 6.2: A vertex-gadget for vertex v . The central node connects to a control path and r receiving paths that can each receive at most one token. For each incident edge e with weight $\omega(e)$, there are $\omega(e)$ relay paths. Each relay path has a first checkpoint (receiving tokens from edge-gadgets) and a second checkpoint (adjacent to the central node, sending tokens to receiving paths). Gray arrows show incoming connections from edge-gadgets.

Thus, for each edge $e = uv \in E(G)$, the i th reservoir node for e is connected via edges to checkpoint 1 of the i th relay path from u to e and the i th relay path from v to e . Additionally, the facilitator node (e, u) is connected via *communication paths* to the guards (u, e) , for each vertex $u \in V(G)$ and incident edge e . See Figure 6.3.

In each of the reductions, for each edge $e = uv \in E(G)$, using their paths, the two facilitator nodes will communicate whether the guards can move their tokens out of the corresponding checkpoint 2 nodes to the receiving paths. In each such edge-gadget, only one facilitator node will be able to instruct its guards to send their tokens, and this is made possible by either the constraint gadget or the budget. This also encodes the fact that each edge can be oriented toward one of its endpoints. It will also be the case that all guards communicating with one facilitator node will send their tokens. In each vertex-gadget, at most r guards can send their tokens to the gadget’s receiving paths, encoding that each vertex can orient at most r units of edge away from itself.

In the INDEPENDENT SET-DISCOVERY reduction, the receiving paths for a vertex $v \in V(G)$ will simply be each made of one node which we call a *receiving node for v* , and will collectively be able to receive r tokens in total, but can also receive less. This perfectly encodes the problem statement that asks for an orientation in which the edges oriented out of a vertex have a total weight of at most r , and not equal to r .

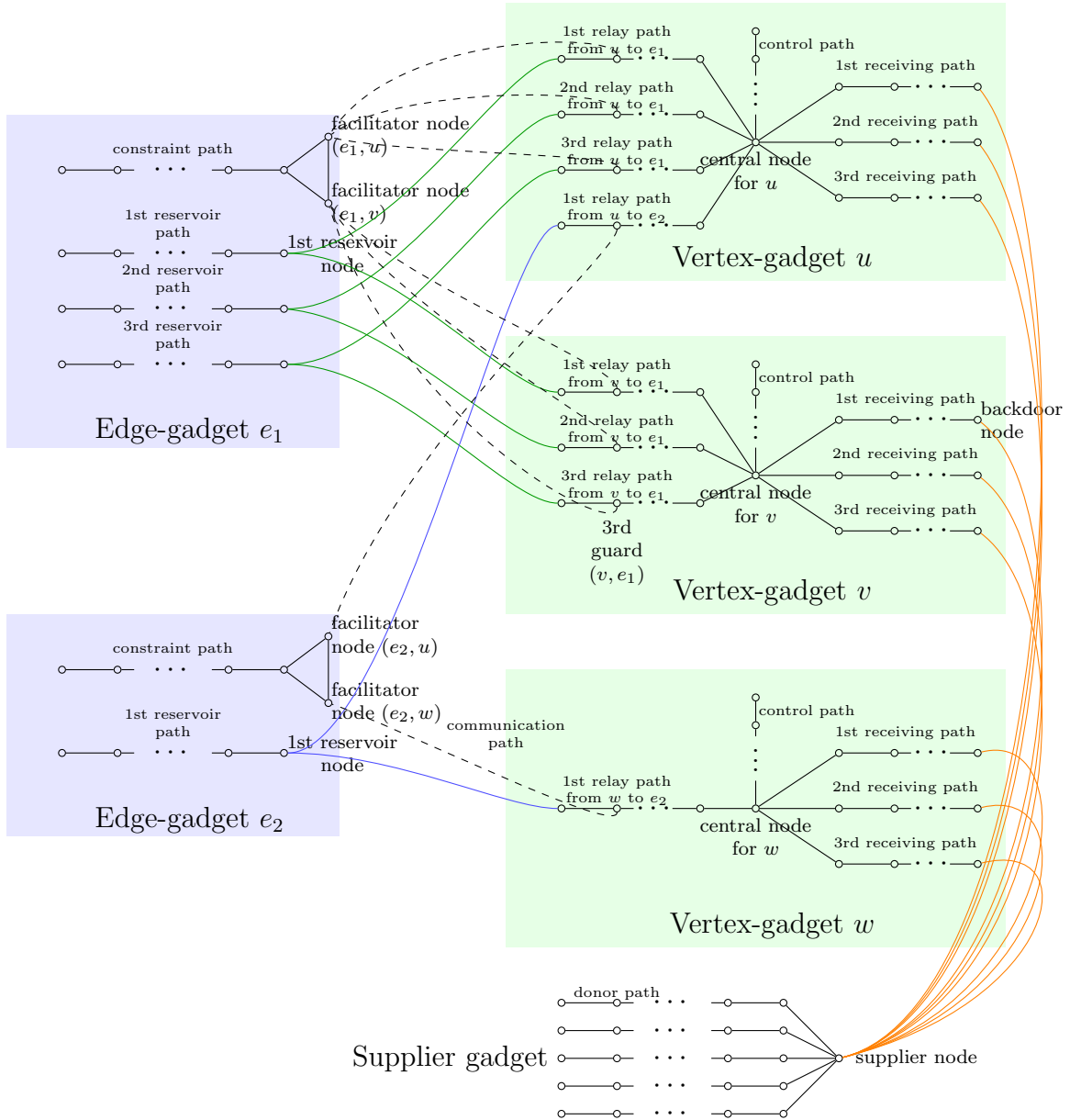


Figure 6.3: Sketch of the graph H constructed by the reduction for an MMO instance with graph G having vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. The supplier gadget with five donor paths ($rn - \Sigma = 5$) is shown at the bottom, with the supplier node adjacent to all backdoor nodes of the receiving paths (through orange edges).

In the other two reductions, the receiving paths for a vertex $v \in V(G)$ will either be edges that need to be covered or paths with some nodes that need to be dominated. Since the instance solution may have some vertices where the total weight of their out-neighbors is less than r , some receiving paths will still need to be “supplied” with a token each to be able to “cover” their edges or “dominate” all their nodes. For this, in those reductions, we will have in H a *supplier gadget*.

The supplier gadget contains a central *supplier node* which connects to $rn - \Sigma$ *donor paths* that can each send one token to the receiving paths. The supplier node connects (via edges) to all receiving paths across all vertex-gadgets through the *backdoor nodes* of their receiving paths. These backdoor nodes are the endpoints of the receiving paths that are not adjacent to the central node, allowing tokens from the donor paths to enter the receiving paths without interfering with the token flow from the relay paths. See Figure 6.3.

Pathwidth of H . Recall that to show that a problem is XNLP-hard, we must perform a parameterized log-space reduction (*pl*-reduction, for short). In such reductions, it is important not only to show that the parameter in the reduced instance remains a function of the parameter in the original instance but also to show that we can perform the reduction using at most $O(h(pw) + \log |x|)$ working space, for an input instance of size $|x|$ and parameter pw , where h is a computable function. In this case, we say that the reduction can be performed by a *log-space transducer*, which can be thought of as an algorithm.

We show that this holds in the following lemma and corollaries.

Lemma 83. *Let $(G, \mathcal{P}_G, \omega, r)$ be an instance of MMO. There exists a log-space transducer that transforms a path decomposition of G to one of H (the graph constructed by the reduction) with width at most $pw + 6$.*

Proof. We build a log-space transducer that, given a path decomposition of G , outputs a path decomposition for the graph H of width at most $pw + 7$. For that, we need the following claim.

Claim 84. *Given a path $P = (v_1, v_2, \dots, v_\ell)$ where ℓ is a constant, a nice path decomposition of P can be constructed with bags where we maintain at most two consecutive vertices at any time:*

- $B_1 = \emptyset$
- $B_2 = \{v_1\}$ (introduce v_1)

- For $i = 2, \dots, \ell$:
 - $B_{2i-1} = \{v_{i-1}, v_i\}$ (introduce v_i)
 - $B_{2i} = \{v_i\}$ (forget v_{i-1})
- $B_{2\ell+1} = \emptyset$ (forget v_ℓ)

This decomposition has width 1 and can be output by a log-space transducer that only needs to track the current position in the path.

Recall that the pathwidth of a graph is equal to the size of the largest bag in the corresponding path decomposition minus 1. Now, given the path decomposition of G , we first ensure that it is a nice path decomposition. A nice path decomposition has the property that it begins and ends with empty bags, and each intermediate bag either introduces or forgets exactly one node. Note that this can be done by a log-space transducer [34].

We then process the bags from left to right. We first output a bag that introduces the supplier node. Then, for each $i \in [rn - \Sigma]$ (where the pointer i needs only $\mathcal{O}(\log n)$ space), we output bags that represent the nice path decomposition of a distinct donor path following the procedure described in Claim 84 and where the supplier node is added to each bag.

For a forget bag in the path decomposition of G , we output a bag containing the central nodes of the vertex-gadgets for the vertices in the forget bag (where the supplier node is also added to each bag). For a bag that introduces a vertex $v \in V(G)$, we output the following bags in order:

1. for $j \in [r + 1]$, we output bags that represent the nice path decomposition of the j th receiving path following the procedure described in Claim 84 and where the supplier node is also added to each bag (these bags have a size larger than the pathwidth of G by only four; one since pathwidth is equal to largest bag size minus one, one from the supplier node, and two from the receiving path decomposition),
2. we output bags that represent the nice path decomposition of the control path for v following the procedure described in Claim 84 and where the supplier node is also added to each bag (these bags have a size larger than the pathwidth of G by only four; one since pathwidth is equal to largest bag size minus one, one from the supplier node, and two from the control path decomposition),
3. for each edge $e \in E(G)$ incident to v with the other endpoint u in the current bag, we do the following in order.

- (a) We output bags that represent the nice path decomposition of the constraint path for e following the procedure described in Claim 84, where the supplier node is also added to each bag, and where we do not forget the node adjacent to the facilitator nodes (e, u) and (e, v) , which we introduce in two following bags. After introducing the facilitator nodes, we forget the node adjacent to them on the constraint path for e . Bags of this step have a size larger than the pathwidth of G by only five (one since pathwidth is equal to largest bag size minus one, one from the supplier node, and three from the facilitator nodes and the node adjacent to them on the constraint path for e).
- (b) For $j \in [\omega(e)]$, we output bags that represent the nice path decomposition of the j th reservoir path following the procedure described in Claim 84, where the supplier node and the facilitator nodes (e, u) and (e, v) are also added to each bag, and where we do not forget the reservoir node. We then output bags that represent the nice path decompositions of the paths that connect the j th reservoir node for e to the checkpoints 2 of the j th relay paths from both u to e and v to e , following the procedure described in Claim 84, where the supplier node and the facilitator nodes (e, u) and (e, v) are also added to each bag. Bags of this step have a size larger than the pathwidth of G by only seven (one since pathwidth is equal to largest bag size minus one, one from the supplier node, and two from the facilitator nodes, one from the reservoir node, and two from the reservoir path decomposition).
- (c) We output bags that forget the facilitator nodes (e, u) and (e, v) .

At the end, we output a bag that forgets the supplier node. The result is a nice path decomposition of the graph H . The width has increased by at most six (since the pathwidth is the largest bag size minus one). $\square$

Note that adding a constant number of nodes c to H cannot increase its pathwidth by more than c or change the fact that we can build it using a log-space transducer. We state this in the following corollary.

Corollary 85. *Let $(G, \mathcal{P}_G, \omega, r)$ be an instance of MMO. Let H' be a graph obtained from H (the graph constructed by the reduction) by adding a constant number of nodes c . There exists a log-space transducer that transforms a path decomposition of G to one of H' with width at most $pw + 6 + c$.*

In all the reductions, the lengths of the constraint, reservoir, relay, control, receiving, and communication paths will be constants that are independent of the input size. Furthermore, b will be bounded by a function that is linear in the input size. This means that

a log-space transducer that outputs H (or H'), along with its path decomposition, the set S and the budget b , need to run only for polynomial time, which combined with the bound on the budget ensures that we obtain a valid pl -reduction.

Corollary 86. *Let $(G, \mathcal{P}_G, \omega, r)$ be an instance of MMO. There exists a log-space transducer that outputs in polynomial time H (or H'), along with its path decomposition of width at most $pw + 6 + c$ for some constant c , the set S and the budget b .*

6.1.2 XNLP-hardness of Independent Set-Discovery for Parameter pathwidth

In this section, we show the following theorem.

Theorem 87. INDEPENDENT SET-DISCOVERY is XNLP-hard when parameterized by pathwidth.

We prove the above the theorem by a reduction from MMO. Given an instance $(G, \mathcal{P}_G, \omega, r)$ of MMO, we construct an instance $(H, \mathcal{P}_H, S, b)$ of INDEPENDENT SET-DISCOVERY. The construction follows the framework from Section 6.1.1 with these modifications. Each receiving path becomes a single *receiving node* and similarly, each control path becomes a single *control node*. Each reservoir path becomes a single *reservoir edge* with a reservoir node and an adjacent *pump node* as endpoints. Each relay path becomes a single *relay edge* between its checkpoint 1 and checkpoint 2. Each constraint path becomes a single *constraint edge* with nodes that we call the *constraint node* and the *decision node*, where the decision node is adjacent to both facilitator nodes. Finally, each communication path between a guard (v, e) (now a checkpoint 2 node) and its corresponding facilitator node (e, v) becomes a single edge that connects them directly. See Figure 6.4.

We place tokens initially on the reservoir and pump nodes of all reservoir paths, on both nodes of the constraint edges, on checkpoint 2 of each relay path, and finally on the control nodes. See Figure 6.4. No supplier gadget is needed since receiving nodes in INDEPENDENT SET-DISCOVERY can remain empty without penalty—the independent set does not require all receiving nodes to contain tokens (they are there to receive extra tokens that cannot be placed elsewhere within the budget).

The high-level idea here is that each token on a reservoir node must slide towards an adjacent checkpoint 1 node, which will trigger the token on the adjacent checkpoint 2 node to slide toward a token-free receiving node. This requires 3Σ token slides in total, where Σ is the total weight of all edges in the MMO instance.

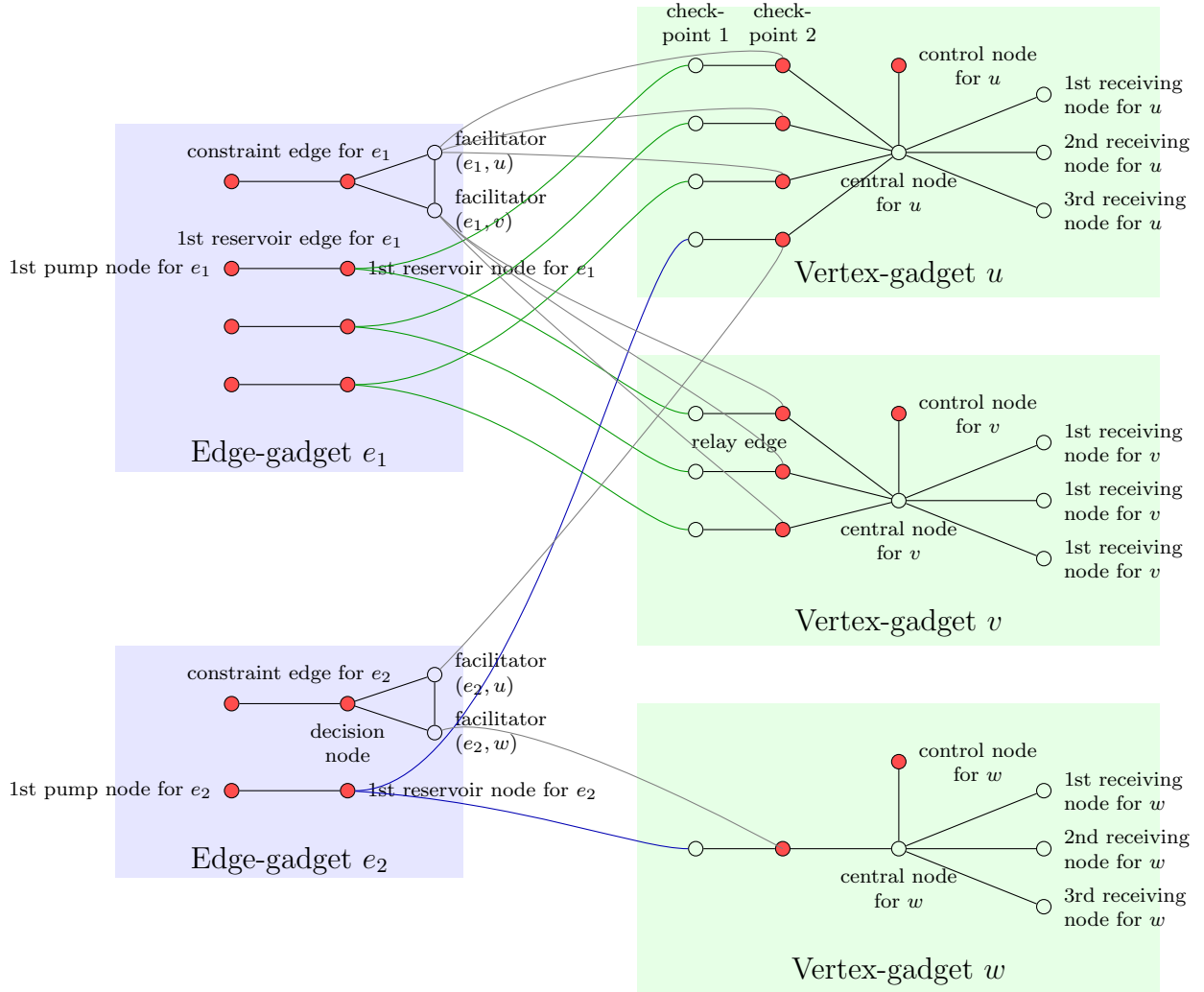


Figure 6.4: The graph H for INDEPENDENT SET-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each edge-gadget has reservoir edges (with reservoir and pump nodes) and a constraint edge. Each vertex-gadget has a central node, control node, receiving nodes, and relay edges connecting from their checkpoint 1 and 2 nodes to the edge-gadgets through edges.

Additionally, each token on a decision node for edge $e = uv \in E(G)$ must slide to one of the adjacent facilitator nodes (e, u) or (e, v) , using one token slide per edge. Our budget b is set to $3\Sigma + m$, which allows exactly these moves. The key constraint is that facilitator

(e, v) is adjacent to all checkpoint 2 nodes of the relay paths from v to e . If the token on the decision node for edge e slides to facilitator (e, v) , these checkpoint 2 nodes must be token-free in the final configuration. This occurs precisely when the tokens on these checkpoint 2 nodes slide to receiving nodes in vertex-gadget v , which in turn happens when tokens on the reservoir nodes for e slide to the relay paths from v to e , thereby encoding that edge e is oriented away from vertex v . We present below the forward and backward directions of the proof of Theorem 87.

Lemma 88. *If $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO, then $(H, \mathcal{P}_H, S, b)$ is a yes-instance of INDEPENDENT SET-DISCOVERY.*

Proof of Lemma 88. Let $\theta : E(G) \rightarrow V(G) \times V(G)$ be an orientation of G such that for each $v \in V(G)$, the total weight of edges directed away from v is at most r .

For each edge $e = uv \in E(G)$ with $\theta(e) = (v, u)$, we do the following.

1. For each $i \in [\omega(e)]$, we slide the token from the reservoir node for the i th reservoir path for e to checkpoint 1 of the i th relay path from v to e (this uses $\omega(e)$ token slides).
2. For each $i \in [\omega(e)]$, we slide the token from checkpoint 2 on the i th relay path from v to e to the central node for v , then to a free receiving node for v (this uses $2\omega(e)$ token slides).
3. We slide the token from the constraint path for e to the facilitator node (e, v) (this uses one token slide).

This totals $m + 3\Sigma$ token slides. Step 2 is possible since θ ensures that each vertex v orients at most weight r away from itself, so at most r tokens need receiving nodes in the vertex-gadget for v . Step 3 is possible since in Step 2 all tokens are removed from the checkpoint 2 nodes of the relay paths from v to e . $\square$

To prove the backward direction, we begin with a simple observation about a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides. Note that in H , it is better to have a token on the control node than on its adjacent central node.

Observation 89. *In a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides, the token on the control node does not slide.*

Lemma 90. *If $(H, \mathcal{P}_H, S, b)$ is a yes-instance of INDEPENDENT SET-DISCOVERY, then $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO.*

Proof of Lemma 90. The minimum number of token slides used inside any constraint gadget for edge $e \in E(G)$ is one, and can be achieved by sliding the token on the decision node to one of the adjacent facilitator nodes. Thus, at least m token slides are required inside the edge-gadgets and the remaining budget is 3Σ . Additionally, the minimum number of token slides required to fix a reservoir edge for edge $e = uv \in E(G)$ and $i \in [\omega(e)]$ is one, and can be achieved by sliding the token on the reservoir node to either checkpoint 1 of the i th relay edge from u to e or checkpoint 1 of the i th relay edge from v to e . This requires Σ token slides. Since a solution that moves the token on the constraint node but not the token on the decision node, or moves the token on the pump node but not the token on the reservoir node, is not minimal, we can safely assume that the described $m + \Sigma$ token slides are executed in any solution that uses the minimum number of token slides.

In these same solutions that use the minimum number of token slides, a token that slides to checkpoint 1 of the i th relay edge from u to e for an edge $e = uv \in E(G)$ and $i \in [\omega(e)]$ requires the token on checkpoint 2 of the same relay edge to slide to either an adjacent facilitator node or the central node for u . This token would thus force one other token slide and in total the Σ tokens that slide to checkpoint 1 nodes thus force Σ other token slides.

A token that slides from a checkpoint 2 node in the vertex-gadget for u to the central node for u must slide again at least once more, since any independent set achieved through the minimal number of slides would never require sliding tokens on control nodes (by Observation 89). Since each constraint gadget contains exactly one facilitator node with a token, a token on a checkpoint 2 node in the vertex-gadget for u that slides to a facilitator node must force at least two more token slides (since the token itself or another token in the constraint gadget must slide to checkpoint 2 nodes that are either already containing tokens, which forces one more token slide, or are token-free because one of their adjacent nodes does, thus also forcing one more token slide).

Given that at most Σ token slides remain in any minimal solution, and that each of the Σ tokens initially on checkpoint 2 nodes that must move uses at least one additional token slide, each such token uses exactly one additional token slide. Thus, the token initially on the checkpoint 2 node of the i th relay edge from u to e that slides to the central node for u , must in exactly one more token slide move to a free receiving node for u . This is possible only if there exists a free receiving node in the vertex-gadget for u .

For any edge $e = uv \in E(G)$, if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration, then no checkpoint 2 node of a relay edge from v to e (resp. from

u to e) contains a token in the final configuration, and the tokens on these checkpoint 2 nodes have been moved to receiving nodes for v (resp. for u). Given this, we produce an orientation θ of G , where $\theta(e) = (v, u)$ (resp. $\theta(e) = (u, v)$) if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration. Since each vertex-gadget for v has exactly r receiving nodes, θ is such that the total weight directed out of any vertex $v \in V(G)$ is at most r . $\square$

6.1.3 XNLP-hardness of Vertex Cover-Discovery for Parameter pathwidth

In this section, we show the following theorem.

Theorem 91. VERTEX COVER-DISCOVERY is XNLP-hard when parameterized by pathwidth.

We prove the above theorem by a reduction from MMO. Given an instance $(G, \mathcal{P}_G, \omega, r)$ of MMO, we construct an instance $(H, \mathcal{P}_H, S, b)$ of VERTEX COVER-DISCOVERY. The construction follows the framework from Section 6.1.1 with these modifications. Each receiving path becomes a single *receiving edge* with a *frontdoor node* and a *backdoor node*. The frontdoor node is the endpoint adjacent to the central node of the vertex-gadget, while the backdoor node is adjacent to the supplier node. Each control path becomes a single *control node*. Each reservoir path becomes a path of two edges: an *auxiliary node* is adjacent to a *pump node* which in turn is adjacent to a reservoir node. Each relay path becomes a single *relay edge* between its checkpoint 1 and checkpoint 2. Each constraint path becomes a single *constraint edge* with nodes that we call the *constraint node* and the *decision node*, where the decision node is adjacent to both facilitator nodes. Each communication path becomes a single edge connecting a checkpoint 2 node directly to its corresponding facilitator node. Each donor path in the supplier gadget becomes a path of two edges. To facilitate the proof for this theorem, we add one node to H and make it adjacent to the supplier node. See Figure 6.5.

We initially place tokens on the pump and reservoir nodes of all reservoir paths, on both nodes of the constraint edges, on checkpoint 2 of each relay edge, on the central nodes, on the supplier node, and on the endpoints of each donor path. We set $b = 3\Sigma + m + 3(rn - \Sigma) = m + 3rn$, where $3(rn - \Sigma)$ token slides will allow one token to slide from each donor path to a distinct receiving edge.

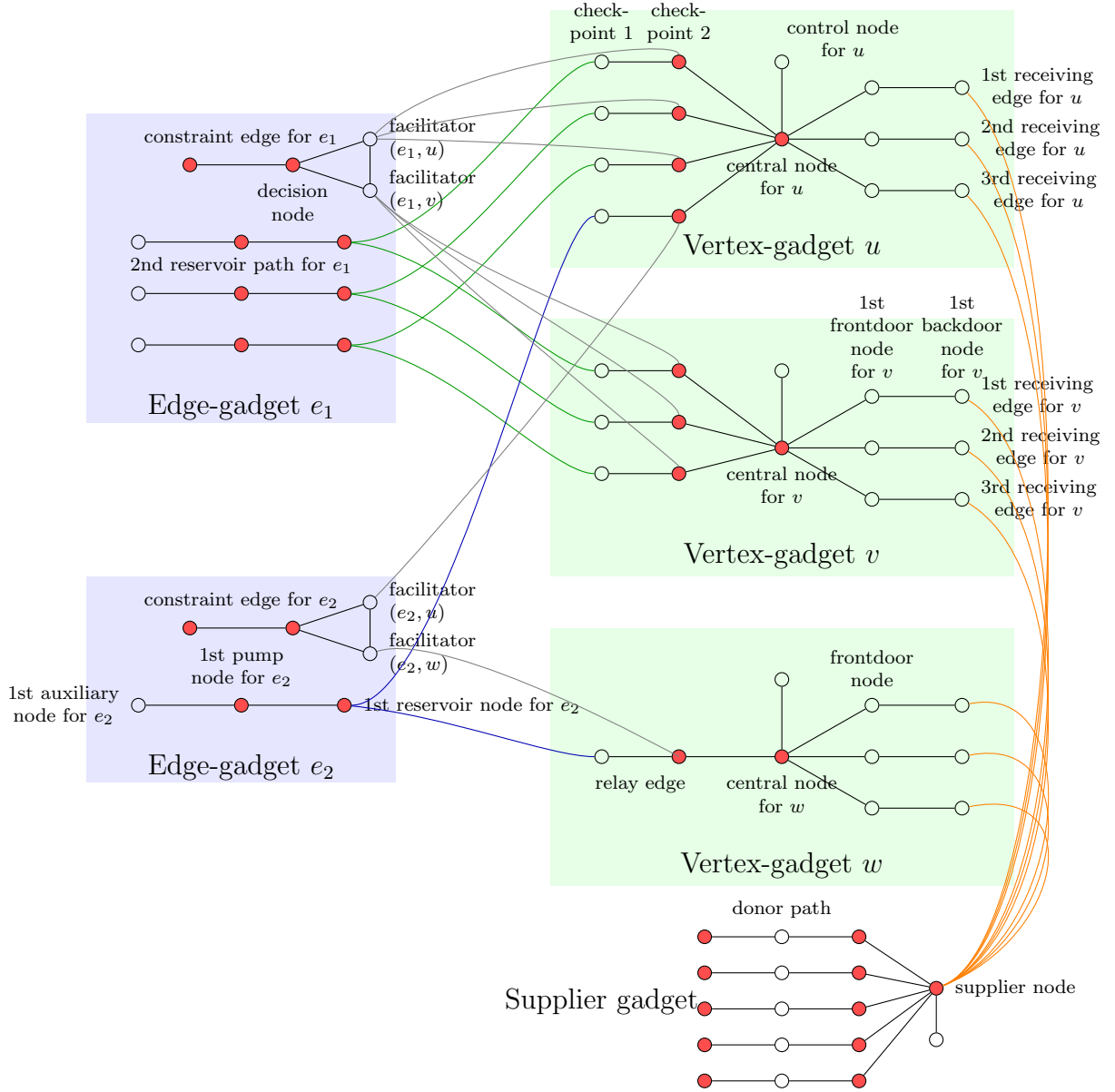


Figure 6.5: The graph H for VERTEX COVER-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each reservoir path has an auxiliary node, a pump node, and a reservoir node. Each receiving edge has one frontdoor node and one backdoor node. The supplier gadget with five donor paths ($rn - \Sigma = 5$) is adjacent to all backdoor nodes.

Unlike in INDEPENDENT SET-DISCOVERY, a supplier gadget is needed because VERTEX COVER-DISCOVERY requires that all edges are covered, including all the receiving edges. Since the instance solution may orient some edges away from vertices with a total weight less than r , some receiving edges will not be covered by tokens from relay edges and must be covered by tokens from donor paths.

The high-level idea is that tokens slide from reservoir nodes to checkpoint 1 nodes, thus allowing the tokens on the adjacent checkpoint 2 nodes to each slide towards a frontdoor node and thus cover one receiving edge. This requires 3Σ token slides. However, note that a token on a checkpoint 2 node can only slide when its adjacent facilitator node contains a token. This forces the choice of one endpoint per edge to orient the edge away from. The construction will force Σ tokens to slide in the described manner to frontdoor nodes. The remaining $rn - \Sigma$ receiving edges will each have to receive a token from a donor path to its backdoor node. We present in the following the forward and backward directions of the proof of Theorem 91.

Lemma 92. *If $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO, then $(H, \mathcal{P}_H, S, b)$ is a yes-instance of VERTEX COVER-DISCOVERY.*

Proof. Let $\theta : E(G) \rightarrow V(G) \times V(G)$ be an orientation of G such that for each $v \in V(G)$, the total weight of edges directed away from v is at most r .

For each edge $e = uv \in E(G)$ with $\theta(e) = (v, u)$, we do the following.

1. For each $i \in [\omega(e)]$, we slide the token from the reservoir node of the i th reservoir path for e to checkpoint 1 of the i th relay edge from v to e (this uses $\omega(e)$ token slides).
2. For each $i \in [\omega(e)]$, we slide the token from checkpoint 2 on the i th relay edge from v to e to the central node for v , then to the frontdoor node of a token-free receiving edge for v (this uses $2\omega(e)$ token slides).
3. We slide the token from the decision node on the constraint edge for e to the facilitator node (e, v) (this uses one token slide).

For each vertex $v \in V(G)$, let w_v denote the total weight of edges oriented away from v under θ . Since $w_v \leq r$, exactly $r - w_v$ receiving edges for v are not covered. For each such receiving edge, we slide a token from the closest endpoint of a donor path through the supplier node to the backdoor node of that receiving edge (this uses two token slides

per donor path). We slide the token on the other endpoint of the path to the middle node of the path (this uses one token slide per donor path).

This totals $m + 3\Sigma + 3(rn - \Sigma) = m + 3rn$ token slides and all edges in H are covered. $\square$

To prove the backward direction, we begin with a simple observation about a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides.

Observation 93. *In a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides, it is safe to assume that the tokens on the central nodes and on the supplier node do not slide.*

Lemma 94. *If $(H, \mathcal{P}_H, S, b)$ is a yes-instance of VERTEX COVER-DISCOVERY, then $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO.*

Proof. First, each receiving edge must be covered by at least one token, and a token can only reach a receiving edge from the central node or the supplier node. This means that we need at least rn token slides to move tokens from either the supplier node or the central nodes and cover the receiving edges.

For a token that is not the token originally on the central node for v to reach this central node, one token slide is needed to slide the token on a checkpoint 2 node in the vertex-gadget for v toward the central node and at least one other token slide is needed to cover the relay edge to which the checkpoint 2 belongs. For a token that is not originally on the supplier node to reach the supplier node, one token slide is needed to slide a token from a donor path and at least one other token slide is needed to cover all edges of this donor path.

Thus, in any case, each receiving edge requires at least three token slides to be covered. Note also that each edge between a pair of adjacent facilitator nodes is not covered and thus at least one additional token slide is required to cover this edge. One token slide can be achieved by sliding the token on the decision node to one of the facilitator nodes and as it is safe to assume that this token slide happens, we do assume as such.

The minimum total number of token slides is thus $b = m + 3rn$, implying that each token that slides to a central node or the supplier node (before sliding to a receiving path) must use or force exactly two token slides.

For a token that is not originally on the supplier node to reach the supplier node, one token slide is needed to slide a token from a donor path and each edge of this donor path can be covered using only one token slide, by sliding the token remaining in the path to the middle node. For a token that moved from a checkpoint 2 node to an adjacent central

node to force at most one more token slide, it can only be the case the adjacent checkpoint 2 node received a token from its adjacent reservoir node. This is true because, if we slide a token from a facilitator node to a checkpoint 2 node instead, we will need at least one more token slide to cover the edge shared between the facilitator node and its adjacent facilitator node.

Using donor paths, we can cover at most $rn - \Sigma$ receiving paths. For each of the Σ other paths, the tokens have to slide from central nodes into frontdoor ones. Note that Σ is exactly equal to the number of reservoir nodes; thus all tokens on all reservoir nodes slide to checkpoint 1 nodes. Now, for edge $e = uv \in E(G)$, only if facilitator (e, v) contains a token, then checkpoint 2 nodes of relay edges from v to e can move their tokens to the central node for v . For each edge $e = uv \in E(G)$, only one facilitator node contains a token, and the tokens on reservoir nodes must all have moved (in one token slide each) to checkpoint 1 nodes of the vertex-gadget for the vertex the facilitator node corresponds to.

We define the orientation $\theta(e) = (v, u)$ if facilitator (e, v) contains a token. Since each vertex v has exactly r receiving edges, and at most r of those edges collectively received one token from each of the relay paths corresponding to edges in G oriented out of v , the weight of edges oriented away from v is at most r . $\square$

6.1.4 XNLP-hardness of Dominating Set-Discovery for Parameter pathwidth

In this section, we show the following theorem.

Theorem 95. DOMINATING SET-DISCOVERY is XNLP-hard when parameterized by pathwidth.

We prove the above theorem by a reduction from MMO. Given an instance $(G, \mathcal{P}_G, \omega, r)$ of MMO, we construct an instance $(H, \mathcal{P}_H, S, b)$ of DOMINATING SET-DISCOVERY. The construction follows the framework from Section 6.1.1 with these modifications. Each receiving path has two edges that connect a *frontdoor node* to a middle node and the middle node to a *backdoor node*. The backdoor nodes are adjacent to the supplier node. Each control path becomes a *control node* adjacent to the central node. Each reservoir path for edge e becomes a reservoir path of four nodes: the *auxiliary node* 1, the *pump node*, the *auxiliary node* 2, and the reservoir node. Each relay path has two edges connecting checkpoint 1 node to the guard node, and the guard node to checkpoint 2 node. Each constraint path has two edges connecting three nodes. Each communication path has two

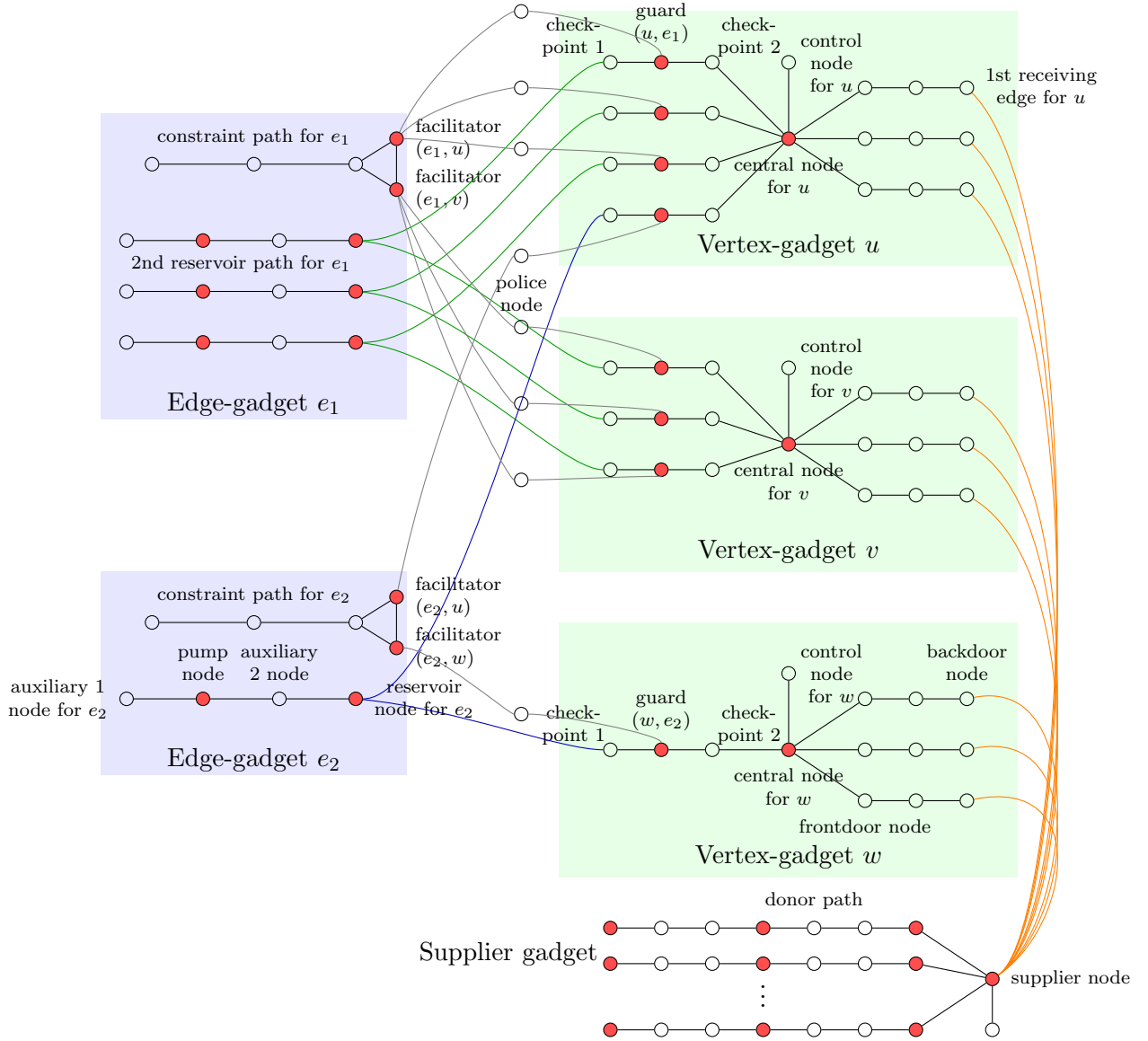


Figure 6.6: The graph H for DOMINATING SET-DISCOVERY constructed from an MMO instance with vertices u, v, w and edges $e_1 = uv$ (weight 3), $e_2 = uw$ (weight 1), with $r = 3$. Red nodes initially contain tokens. Each reservoir path has four nodes. Each relay path has a guard node between checkpoints. Communication paths have police nodes. Receiving paths have one frontdoor, one middle, and one backdoor node. The supplier gadget with five donor paths (each with seven nodes) is adjacent to all backdoor nodes. The edge added to H with a dominator node is omitted but connects to all checkpoint 2 nodes.

edges with one *police node* in the middle. Each donor path is a path with seven nodes. To facilitate the proof of the theorem, we add one node to H and make it adjacent to the supplier node. Additionally, we add to H an edge with a *dominator node* which is made adjacent to all the checkpoint 2 nodes. See Figure 6.6.

We initially place tokens on the pump and reservoir nodes of all reservoir paths, on both facilitator nodes, on the guard nodes of each relay edge, on the central nodes, on the supplier node, on the dominator node, and on the endpoints and the middle node of each donor path. We set $b = 4\Sigma + 2m + 4(rn - \Sigma) = m + 4rn$, where $4(rn - \Sigma)$ token slides will allow tokens to slide from donor paths to receiving edges.

Unlike in INDEPENDENT SET-DISCOVERY, a supplier gadget is needed because DOMINATING SET-DISCOVERY requires that all nodes be dominated, including all the nodes of the receiving paths. Since the instance solution may orient some edges away from vertices with a total weight less than r , some nodes on the receiving paths will not be dominated by tokens from relay edges and must be dominated by tokens from donor paths.

The high-level idea is that (Σ) tokens slide from reservoir nodes to checkpoint 1 nodes, thus allowing the tokens on the adjacent guard nodes to slide (using three token slides each) toward a frontdoor node and thus dominate the middle node of a receiving path. This requires 4Σ token slides. However, note that tokens on guard nodes can only slide when their adjacent police nodes are dominated by a token on a facilitator node. The constraint path requires that one token on one of the facilitator nodes slide towards it. This forces the choice of one endpoint per edge to orient the edge away from. The construction will force Σ tokens to slide in the described manner to frontdoor nodes. The remaining $rn - \Sigma$ receiving paths will each have to receive a token from a donor path to its backdoor node. We present in the following the forward and backward directions of the proof of Theorem 95.

Lemma 96. *If $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO, then $(H, \mathcal{P}_H, S, b)$ is a yes-instance of DOMINATING SET-DISCOVERY.*

Proof. Let $\theta : E(G) \rightarrow V(G) \times V(G)$ be an orientation of G such that for each $v \in V(G)$, the total weight of edges directed away from v is at most r .

For each edge $e = uv \in E(G)$ with $\theta(e) = (v, u)$, we do the following.

1. For each $i \in [\omega(e)]$, we slide the token from the reservoir node of the i th reservoir path for e to checkpoint 1 of the i th relay edge from v to e (this uses $\omega(e)$ token slides).

2. For each $i \in [\omega(e)]$, we slide the token from the guard node on the i th relay path from v to e to the central node for v through a checkpoint 2 node, then to the frontdoor node of a token-free receiving path for v (this uses $3\omega(e)$ token slides).
3. We slide the token from facilitator node (e, u) to the middle node of the constraint path for e (this uses two token slides).

For each vertex $v \in V(G)$, let w_v denote the total weight of edges oriented away from v under θ . Since $w_v \leq r$, exactly $r - w_v$ receiving paths for v have middle nodes that are not dominated. For each such receiving path, we slide a token from the closest endpoint of a donor path through the supplier node to the backdoor node of that receiving path (this uses two token slides per donor path) and dominate all nodes of the receiving path (front door nodes are dominated by the token on the central node). We slide each of the other tokens of the donor path one token slide in the direction of the supplier node to fix the path (this uses two token slides per donor path).

This totals $2m + 4\Sigma + 4(rn - \Sigma) = m + 4rn$ slides and all nodes in H are dominated. $\square$

To prove the backward direction, we begin with a simple observation about a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides.

Observation 97. *In a solution to a reduced $(H, \mathcal{P}_H, S, b)$ instance that requires the minimum number of token slides, it is safe to assume that the tokens on the central nodes and on the supplier node do not slide.*

Lemma 98. *If $(H, \mathcal{P}_H, S, b)$ is a yes-instance of DOMINATING SET-DISCOVERY, then $(G, \mathcal{P}_G, \omega, r)$ is a yes-instance of MMO.*

Proof. First, each receiving path must receive at least one token, and tokens can only reach a receiving path after passing through the central node or the supplier node. This means that we need at least three token slides to move a token through a central node, and at least two token slides to move a token through a supplier node.

For a token to slide through the central node from a guard node, it must be the case that in a valid final configuration T either the guard node or the adjacent checkpoint 1 node contains a token. This implies that for each such token that slides through the central node from a guard node, we need at least one additional token slide and exactly one token slide is only possible by sliding the token on the reservoir node adjacent to the checkpoint 1 node to the checkpoint 1 node. For a token to slide through a supplier node from the closest endpoint of a donor path, it must be the case that we need at least two additional

token slides to fix the donor path and exactly two token slides can only be achieved by sliding each other token on the same donor path one token sliding step in the direction of the supplier node.

Thus, in any case, each receiving path requires at least four token slides (from the analysis above) to dominate its middle nodes. Note also that each constraint path requires at least two token slides to be dominated by sliding a token from one of the facilitator nodes (in the same constraint gadget) to its middle node. The minimum total number of token slides is thus $b = 2m + 4rn$, implying that for each edge-gadget, exactly one facilitator node is in T , and each of the receiving paths forces exactly four token slides.

Using donor paths with at most four token slides per receiving path, we can dominate at most $rn - \Sigma$ middle nodes of $rn - \Sigma$ such paths; for each of the Σ other paths, the tokens have to slide from central nodes into frontdoor ones. Note that this is exactly equal to the number of reservoir nodes, thus all tokens on all reservoir nodes slide to checkpoint 1 nodes. Now, for edge $e = uv \in E(G)$, only if facilitator (e, v) contains a token, then guard nodes of relay paths from v to e can move their tokens to the central node for v , otherwise, the adjacent police node will not be dominated. For each edge $e = uv \in E(G)$, only one facilitator node contains a token, and the tokens on the reservoir nodes of the edge-gadget for e must all have moved (in one token slide each) to checkpoint 1 nodes of the vertex-gadget for the vertex the facilitator node corresponds to (u if the facilitator node is facilitator (e, u) , v otherwise).

We define the orientation $\theta(e) = (v, u)$ if facilitator (e, v) contains a token. Since each vertex v has exactly r receiving paths, and at most r of those paths received tokens from each of the relay paths corresponding to edges in G oriented out of v , the weight of edges oriented away from v is at most r . $\square$

6.2 No Polynomial Kernel for Independent Set-Discovery, Vertex Cover-Discovery, and Dominating Set-Discovery with Parameter $b + \text{pathwidth}$

In this section, we show that all of INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY parameterized by $b + \text{pathwidth}$ do not admit polynomial kernels unless $\text{NP} \subseteq \text{coNP}/\text{P}$. We achieve this by providing an or-cross-composition for each problem from the MINIMUM MAXIMUM OUTDEGREE (MMO) problem. Our or-cross-composition builds upon the reductions used to establish XNLP-hardness,

extending them to handle multiple instances simultaneously while ensuring that the or-cross-composed instance has a solution if and only if at least one of the input instances does. We provide, as before, a framework section that describes the common features of all or-cross-compositions.

6.2.1 The or-cross-composition framework

Let $\mathcal{R}$ be the polynomial equivalence relation over instances of MMO where two instances are equivalent if and only if their graphs have the same number of vertices n , the same number of edges m , the same total weight on the edges (which we denote by Σ) and the same threshold value r . This relation can be decided in polynomial time and partitions MMO instances into classes based on their number of vertices, number of edges, total weight on the edges, and thresholds.

Given t equivalent MMO instances $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ for $j \in [t]$, each with n vertices, m edges, total weight Σ on the edges and a threshold $r^j = r$, we construct a graph H^+ that encodes all instances while enforcing that exactly one can be selected and solved.

The graph H^+ will encode each of the t MMO instances in a subgraph H^j that is built as we built the graph H in Section 6.1.1, and will have the following additional components of an *instance selector* that enforces that exactly one of the t instances is selected for solving.

The instance selector contains for each instance $j \in [t]$, a *selection edge for j* with a *select node for j* and an *unselect node for j* as endpoints. The placement of tokens on these edges will determine which instance is active. The instance selector also contains a *weights-hub component* formed of a *weights-hub node* adjacent to Σ *weight edges*. For each weight edge, we call the endpoint adjacent to the weights-hub node a *weight node*. Finally, the instance selector contains an *orientations-hub component* with an *orientations-hub node* adjacent to m *orientation edges*. For each orientation edge, we call the endpoint adjacent to the orientations-hub node an *orientation node*. These edges will encode the orientation decisions for the m edges in the selected instance.

The instance selector will enforce the OR behavior through careful budget constraints and structural design. It will be attached to each subgraph H^j for $j \in [t]$ only through its weights-hub and orientations-hub nodes. Additionally, each selection edge for j will connect to subgraph H^j . It is easy to see that the graph H^+ can be formed in time polynomial in the size of all t input instances. Additionally, b will be set in each or-cross-composition so that it is bounded above by a function linear in m and Σ . In addition to correctness and the fact that this or-cross-composition runs in polynomial time with respect to the total size of all t instances of MMO, it remains to show that the pathwidth of H^+ is also

bounded above by a function polynomial in the size of the maximum size instance plus $\log t$.

Claim 99. *The pathwidth of H^+ is at most $pw + 10 + c$ for some constant c and where pw is the pathwidth of the maximum pathwidth instance.*

Proof of Claim 99. If we remove the weights-hub and the orientations-hub nodes from H^+ , we are left with the following maximal connected components: Σ weight edges (of pathwidth at most two each) and m orientation edges (of pathwidth at most two each) as well as t subgraphs (each representing an instance) of pathwidth at most $pw + 8 + c$ for some constant c (an extra two to account for the selection edge connected to the subgraph of its corresponding instance). Then an algorithm that outputs a path decomposition for H^+ can simply output the path decomposition for each of the connected components following the procedures described in Claim 84 and Lemma 83, except without empty bags, and where each bag in the decomposition is augmented by the weights-hub and the orientations-hub nodes. The decomposition starts by introducing these nodes and ends by forgetting them. $\square$

Note that adding a polynomial (with respect to m and Σ) number of degree-1 nodes to H^+ (which we would need to do in some or-cross-compositions) cannot increase its pathwidth by more than one. We state this in the following corollary.

Corollary 100. *Let $H^{+'}$ be a graph obtained from H^+ by adding a polynomial (with respect to m and Σ) number of degree-1 nodes. Then its pathwidth is at most $pw + 11 + c$ for some constant c and where pw is the pathwidth of the maximum pathwidth instance.*

6.2.2 No Polynomial Kernel for Independent Set-Discovery with Parameter $b + \text{pathwidth}$

In this section, we show that INDEPENDENT SET-DISCOVERY when parameterized by $b + \text{pathwidth}$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/P$. Thus, we prove the following theorem.

Theorem 101. *There exists an or-cross-composition from MINIMUM MAXIMUM OUT-DEGREE to INDEPENDENT SET-DISCOVERY parameterized by $b + \text{pathwidth}$. Consequently, INDEPENDENT SET-DISCOVERY does not admit a polynomial kernel with respect to $b + \text{pathwidth}$, unless $\text{NP} \subseteq \text{coNP}/P$.*

We prove the above theorem by or-cross-composing MMO into INDEPENDENT SET-DISCOVERY. Given t equivalent MMO instances $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ for $j \in [t]$, each with n vertices, m edges, Σ total weight on the edges, and a threshold $r^j = r$, we construct a graph H^+ following the framework of Section 6.2.1, where the subgraph H^j representing instance $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is as described in Section 6.1.2, and where we add two degree-1 nodes. In particular,

- each receiving path becomes a single *receiving node*, and each control path becomes a single *control node*,
- each reservoir path becomes a single *reservoir edge* with a reservoir node and an adjacent *pump node* as endpoints,
- each relay path becomes a single *relay edge* between its checkpoint 1 and checkpoint 2 endpoints,
- each constraint path becomes a single *constraint edge* with nodes that we call the *constraint node* and the *decision node*, where the decision node is adjacent to both facilitator nodes, and
- each communication path between a guard (v, e) (now a checkpoint 2 node) and its corresponding facilitator node (e, v) becomes a single edge that connects them directly.

Additionally, we make the orientations-hub node adjacent to every constraint node in every subgraph H^j for $j \in [t]$ and to one degree-1 *first extra node*. We make the weights-hub node adjacent to every pump node in every subgraph H^j for $j \in [t]$ and to one degree-1 *second extra node*. Finally, for each $j \in [t]$, we make the unselect node for j adjacent to all the reservoir and central nodes of H^j s. See Figure 6.7.

We place tokens on all reservoir, control, decision, and checkpoint 2 nodes. Additionally, we place a token on each of unselect node for j for each $j \in [t]$, the first and second extra nodes, and all the nodes of the weight and orientation edges.

The high-level idea is that now the tokens that were initially on the pump and constraint nodes of a reduced instance and that forced the tokens on reservoir and decision nodes to slide are outside of the instance on orientation and weight nodes. We set the budget tightly so that in addition to the $m + 3\Sigma$ token slides needed inside some subgraph H^j for $j \in [t]$ and the $2m + 2\Sigma$ token slides needed to slide the tokens on the orientation and weight nodes toward the constraint and pump nodes, respectively, only one additional token slide

is allowed. This token slide can only be used to slide the token on the unselect node for j so that subgraph H^j can receive additional tokens. Thus, we set $b = 3m + 5\Sigma + 1$.

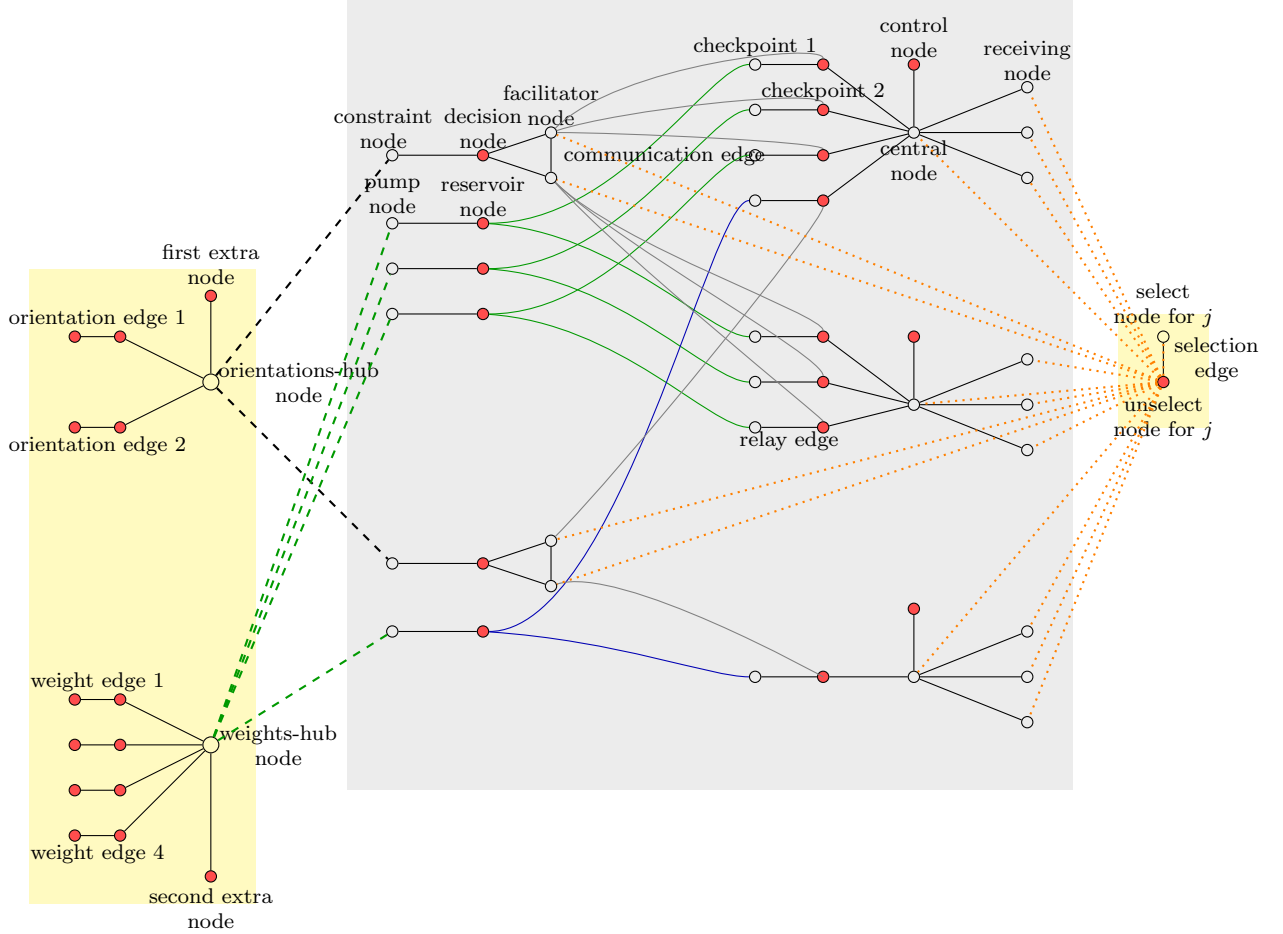


Figure 6.7: Part of the instance selector of the graph H^+ of INDEPENDENT SET-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. The orientations-hub node is adjacent to all constraint nodes (black dashed edges), the weights-hub node is adjacent to all pump nodes (green dashed edges), and the unselect node for j is adjacent to all the reservoir and central nodes in H^j (orange dotted edges).

We now show the forward and backward directions of the proof of Theorem 101.

Lemma 102. *If, for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO, then (H^+, S, b) is a yes-instance of INDEPENDENT SET-DISCOVERY.*

Proof of Lemma 102. Let $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ be a yes-instance of MMO and let $\theta : E(G^j) \rightarrow V(G^j) \times V(G^j)$ be an orientation of G^j such that for each $v \in V(G^j)$, the total weight of edges directed away from v is at most r^j . In H^+ , one token needs to leave each of the orientation nodes and the weight nodes. Thus, we slide the token on each orientation node toward a distinct constraint node in H^j and the token on each weight node toward a distinct pump node in H^j using $2m + 2\Sigma$ token slides. Additionally, we slide the token on the unselect node for j toward the select node for j in one token slide. This leaves a budget $b' = m + 3\Sigma$ and achieves a configuration of tokens inside H^j that is equal to that of the tokens on the graph H of the reduction of Theorem 87. From Lemma 88, given θ , we can achieve a configuration of the tokens that constitutes an independent set in H^j in $m + 3\Sigma$ token slides. Applying these token slides in H^+ also achieves a configuration of tokens that is an independent set. $\square$

Before showing the backward direction, we note the following two observations.

Observation 103. *In a solution to an instance (H^+, S, b) of INDEPENDENT-SET-DISCOVERY that requires the minimum number of token slides, it is safe to assume that the tokens on the control nodes and on the first and second extra nodes do not slide.*

Observation 104. *In a solution to an instance (H^+, S, b) of INDEPENDENT-SET-DISCOVERY that requires the minimum number of token slides, no token can slide into a subgraph H^j for some $j \in [t]$ without sliding the token on its corresponding unselect node for j .*

Lemma 105. *If (H^+, S, b) is a yes-instance of INDEPENDENT SET-DISCOVERY, then for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO.*

Proof of Lemma 105. First, from Observation 103 we can conclude that one token has to slide from each orientation edge two token slides towards some constraint node. We also conclude that one token has to slide from each weight edge two token slides towards some pump node. Furthermore, from Observation 104, we also conclude that at least one token slide is needed to slide the token on the unselect node for j for some $j \in [t]$. Note that we can safely assume that for some solution that uses the minimum number of token slides, this token goes to the select node for j .

This implies that at most $m + 3\Sigma$ token slides are left. Each token on a constraint node in an edge-gadget for some edge e in some subgraph H^j for some $j \in [t]$ (and many

tokens can be on this node) will force at least one additional token slide and a token to slide towards one of the facilitator nodes in the same gadget. Additionally, each token on a pump node (and many tokens can be on this node) will force at least one additional token slide and a token to slide from the adjacent reservoir node one token sliding step toward a checkpoint 1 node. This leaves at most 2Σ token slides and Σ tokens at checkpoint 1 nodes.

Each of those Σ tokens will require at least two token slides. This is true because a token sliding from an adjacent checkpoint 2 node cannot be destined to a central node by Observation 103, and if it slides into a facilitator, then it requires at least one additional token slide, since each triangle formed by a decision and two facilitator nodes contains at least one token. Thus, each of these Σ tokens can use exactly two token slides. Given that this consumes the budget and thus we can afford to move only one token on one unselect node, all tokens moving from orientation and weight edges must go to the same subgraph H^j for some $j \in [t]$.

In fact, in a solution that requires the minimum number of token slides, each token that slides from a checkpoint 2 node into a facilitator node requires at least two additional token slides as it must be the case that it or some other token leaves toward some other checkpoint 2 node that either contains a token or has an adjacent checkpoint 1 node that does. Thus, each token that leaves a checkpoint 2 node goes to a central node and then a receiving node.

Given that this consumes all of the budget, it must be the case that the tokens that come from the orientation edges slide into distinct constraint nodes of H^j and from the weight edges into distinct pump nodes of H^j and then they force the tokens on adjacent nodes to slide (otherwise, we would need more token slides to slide at least one token along a constraint or a reservoir edge). Additionally, each token that slides into a checkpoint 1 node will make the token on the adjacent checkpoint 2 node slide two token slides toward a receiving node and each token that slides into a constraint node will make the token on the decision node slide toward an adjacent facilitator node.

For any edge $e = uv \in E(G^j)$, if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration, then no checkpoint 2 node of relay edges from v to e (resp. from u to e) contains a token in the final configuration, and the tokens on these checkpoint 2 nodes have been moved to receiving nodes for v (resp. for u). Given this, we produce an orientation θ of G^j , where $\theta(e) = (v, u)$ (resp. $\theta(e) = (u, v)$) if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration. Since each vertex-gadget for v has exactly r receiving nodes, θ is such that the total weight directed out of any vertex $v \in V(G)$ is at most r . $\square$

6.2.3 No Polynomial Kernel for Vertex Cover-Discovery with Parameter $b+\text{pathwidth}$

In this section, we show that VERTEX COVER-DISCOVERY when parameterized by $b+\text{pathwidth}$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/P$. Thus, we prove the following theorem.

Theorem 106. *There exists an or-cross-composition from MINIMUM MAXIMUM OUT-DEGREE to VERTEX COVER-DISCOVERY parameterized by $b+\text{pathwidth}$. Consequently, VERTEX COVER-DISCOVERY does not admit a polynomial kernel with respect to $b+\text{pathwidth}$, unless $\text{NP} \subseteq \text{coNP}/P$.*

We prove the above theorem by or-cross-composing MMO into VERTEX COVER-DISCOVERY. Given t equivalent MMO instances $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ for $j \in [t]$, each with n vertices, m edges, Σ total weight on the edges, and a threshold $r^j = r$, we construct a graph H^+ following the framework of Section 6.2.1, where the subgraph H^j representing instance $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is as described in Section 6.1.3 except without the supplier gadget, and where we add a polynomial (with respect to m and Σ) number of degree-1 nodes. In particular,

- each receiving path becomes a single *receiving edge* with a *frontdoor node* and *backdoor node* as endpoints (the frontdoor node is the endpoint adjacent to the central node of the vertex-gadget),
- each control path becomes a single *control node*,
- each reservoir path becomes a path of two edges: an *auxiliary node* is adjacent to a *pump node* which in turn is adjacent to a reservoir node,
- each relay path becomes a single *relay edge* between its checkpoint 1 and checkpoint 2 endpoints,
- each constraint path becomes a single *constraint edge* with nodes that we call the *constraint node* and the *decision node*, where the decision node is adjacent to both facilitator nodes, and
- each communication path between a guard (v, e) (now a checkpoint 2 node) and its corresponding facilitator node (e, v) becomes a single edge that connects them directly.

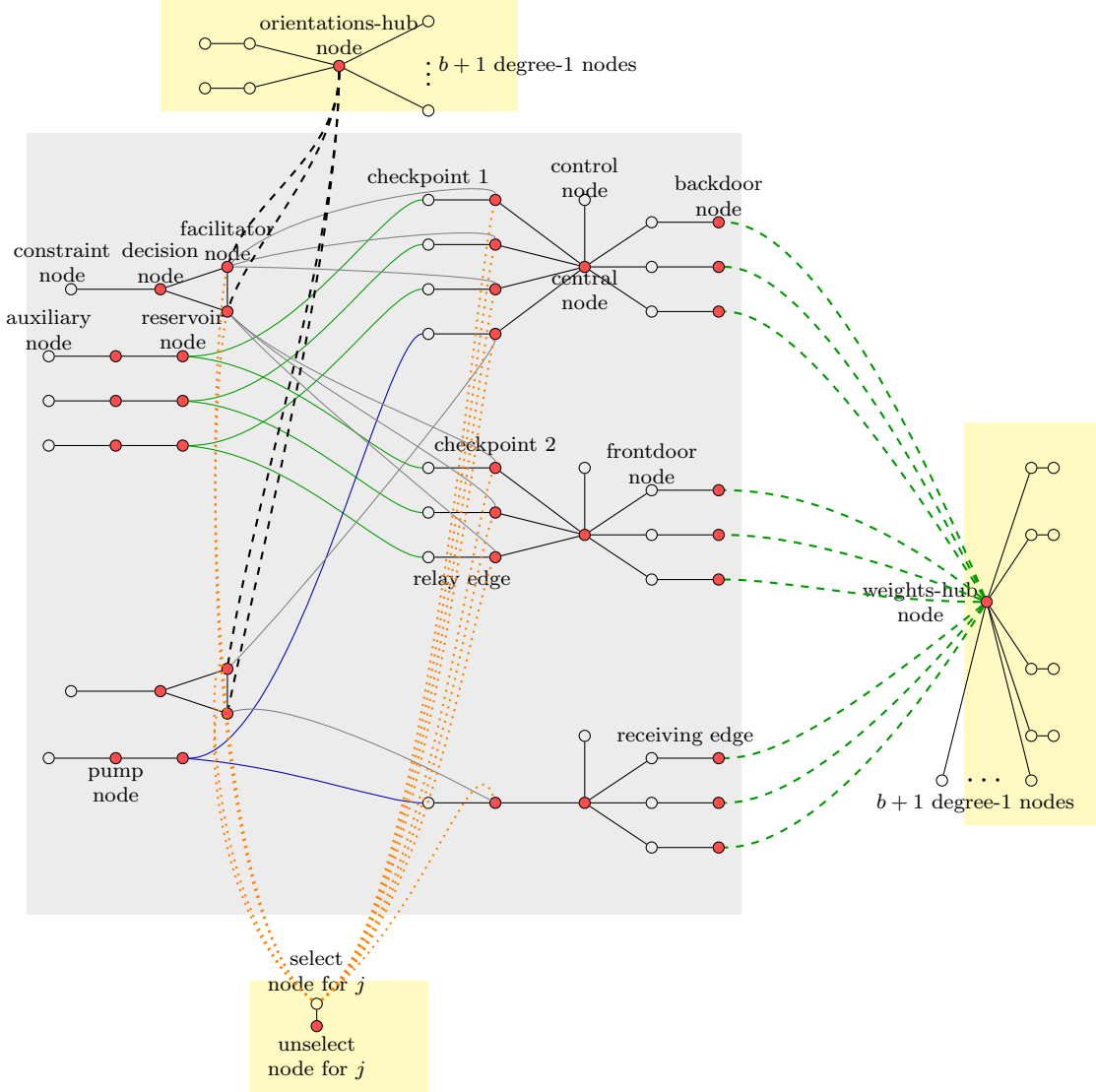


Figure 6.8: Part of the instance selector of the graph H^+ of VERTEX COVER-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. Each hub node is adjacent to $b + 1$ pendant nodes. The orientations-hub node is adjacent to all facilitator nodes (black dashed edges), the weights-hub node is adjacent to all backdoor nodes of receiving edges (green dashed edges), and the select node for j is adjacent to all facilitator and checkpoint 2 nodes in H^j (orange dotted edges).

Additionally, we make the orientations-hub node adjacent to every facilitator node and the weights-hub node adjacent to every backdoor node in every subgraph H^j for $j \in [t]$. Finally, for each $j \in [t]$, we make the select node for j adjacent to all the facilitator and checkpoint 2 nodes of H^j s. See Figure 6.8.

We place tokens on all reservoir, pump, central, decision, facilitator, backdoor, and checkpoint 2 nodes. Additionally, we place a token on each unselect for j for each $j \in [t]$, and the weights-hub and orientations-hub nodes.

The high-level idea is that now each weight edge needs at least one token to cover its edge and will have to take this token from a backdoor node. Additionally, each orientation edge needs at least one token to cover its edge and will have to take this token from a facilitator node. This will consume $2\Sigma + 2m$ token slides. We tighten the budget so that in addition to these token slides, only $3\Sigma + 1$ token slides are allowed. One token slide can be used to slide the token on the unselect node for j toward the select node for j which covers edges this node shares with the subgraph that loses tokens. The other 3Σ token slides can be used to fill frontdoor nodes adjacent to now empty backdoor nodes, thus ensuring that we move tokens from distinct backdoor nodes and from the same subgraph H^j . Thus, $b = 2m + 5\Sigma + 1$.

We now show the forward and backward directions of the proof of Theorem 106.

Lemma 107. *If, for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO, then (H^+, S, b) is a yes-instance of VERTEX COVER-DISCOVERY.*

Proof of Lemma 107. Let $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ be a yes-instance of MMO and let $\theta : E(G^j) \rightarrow V(G^j) \times V(G^j)$ be an orientation of G^j such that for each $v \in V(G^j)$, the total weight of edges directed away from v is at most r^j . In H^+ , for each edge $e \in G^j$ such that $\theta(e) = (v, u)$, we slide the token on facilitator (e, u) into a token-free orientation node. This consume $2m$ token slides. We slide the tokens on the reservoir nodes in the edge-gadget for e into their adjacent checkpoint 1 nodes in the vertex-gadget for v . This requires Σ token slides.

Now, we slide each token on a checkpoint 2 node adjacent to a checkpoint 1 node containing a token into a token-free frontdoor node in the same gadget. Since this is a feasible orientation, each vertex-gadget receives at most r tokens on its checkpoint 1 nodes, thus r tokens can slide from checkpoint 2 nodes of a vertex-gadget. This requires 2Σ token slides.

Since the number of tokens now on distinct frontdoor nodes is Σ , we slide each token on a backdoor node adjacent to a frontdoor node containing a token to a distinct token-free

weight node. This requires 2Σ token slides. Finally, we slide the token on the unselect node for j into the select node for j using one token slide. Performing these $b = 2m + 5\Sigma + 1$ token slides results in a vertex cover in H^+ . $\square$

Before showing the backward direction, we note the following two observations.

Observation 108. *In a solution to an instance (H^+, S, b) of VERTEX COVER-DISCOVERY that requires the minimum number of token slides, it is safe to assume that the tokens on the central nodes and on the orientations-hub and weights-hub nodes do not slide.*

Observation 109. *In a solution to an instance (H^+, S, b) of VERTEX COVER-DISCOVERY that requires the minimum number of token slides, no token can slide from a subgraph H^j for some $j \in [t]$ without sliding a token into its corresponding select node for j .*

Lemma 110. *If (H^+, S, b) is a yes-instance of VERTEX COVER-DISCOVERY, then for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO.*

Proof of Lemma 110. First, from Observation 108 we can conclude that Σ tokens have to slide from backdoor nodes into weight nodes. We also conclude that m tokens have to slide from facilitator nodes to orientation nodes. This requires $2m + 2\Sigma$ token slides. Furthermore, from Observation 109, we also conclude that at least one token slide is needed to slide a token into the select node for j for some $j \in [t]$. Note that we can safely assume that for some solution that uses the minimum number of token slides, this token comes from the select node for j .

This implies that at most 3Σ token slides are left. Each token that slides from a backdoor node requires at least two token slides to get a token to an adjacent frontdoor node, since by Observation 108 the token cannot come from a central node. Also, when a token moves from a checkpoint 2 node, it requires at least one token slide to slide a token into either the same node or an adjacent checkpoint 1 node. Note that this consumes the entire budget, thus when a token slides from a checkpoint 2 node it can force exactly one token slide. Additionally, given the tightness of the budget, it must be the case that tokens move into orientation and weight nodes from the same subgraph H^j for some $j \in [t]$. Moreover, it must be the case that these tokens come from distinct facilitator nodes in m distinct edge-gadgets of H^j into the orientation nodes. Otherwise, we would need at least one additional token slide to cover the edge shared by two adjacent facilitator nodes or to get a token into a facilitator node.

Note that since every constraint gadget in H_j is missing a token on one of its facilitator nodes, when a token moves from a checkpoint 2 node cp , it must be the case that a token

slides from a reservoir node into the checkpoint 1 node adjacent to cp . Otherwise, we would require at least one additional token slide. Since Σ tokens slide into the weight nodes from backdoor nodes, Σ tokens slide from the reservoir nodes. Thus, each edge-gadget slides all the tokens on its reservoir nodes.

For any edge $e = uv \in E(G^j)$, only if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration, then a checkpoint 2 node of relay edges from v to e (resp. from u to e) can not contain a token in the final configuration. Thus, so that Σ tokens slide from reservoir nodes, all tokens on the reservoir nodes in the edge-gadget for e slide to the checkpoint 1 nodes of the vertex-gadget for v . However, since each vertex-gadget should receive at most r such tokens and consequently send at most r tokens to the weight edges, we can produce a feasible orientation θ of G^j as follows. We set $\theta(e) = (v, u)$ (resp. $\theta(e) = (u, v)$) if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration. Since each vertex-gadget for v can send at most r tokens into weight nodes, and thus receive at most r tokens into its checkpoint 1 nodes, θ is such that the total weight directed out of any vertex $v \in V(G)$ is at most r . $\square$

6.2.4 No Polynomial Kernel for Dominating Set-Discovery with Parameter $b+\text{pathwidth}$

In this section, we show that DOMINATING SET-DISCOVERY when parameterized by $b+\text{pathwidth}$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/P$. Thus, we prove the following theorem.

Theorem 111. *There exists an or-cross-composition from MINIMUM MAXIMUM OUT-DEGREE to DOMINATING SET-DISCOVERY parameterized by $b+\text{pathwidth}$. Consequently, DOMINATING SET-DISCOVERY does not admit a polynomial kernel with respect to $b+\text{pathwidth}$, unless $\text{NP} \subseteq \text{coNP}/P$.*

We prove the above theorem by or-cross-composing MMO into DOMINATING SET-DISCOVERY. Given t equivalent MMO instances $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ for $j \in [t]$, each with n vertices, m edges, Σ total weight on the edges, and a threshold $r^j = r$, we construct a graph H^+ following the framework of Section 6.2.1, where the subgraph H^j representing instance $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is as described in Section 6.1.4 except without the supplier gadget and the dominator node. In particular,

- each receiving path has two edges that connect a *frontdoor node* to a middle node and the middle node to a *backdoor node* (the frontdoor node is the endpoint adjacent to the central node of the vertex-gadget),

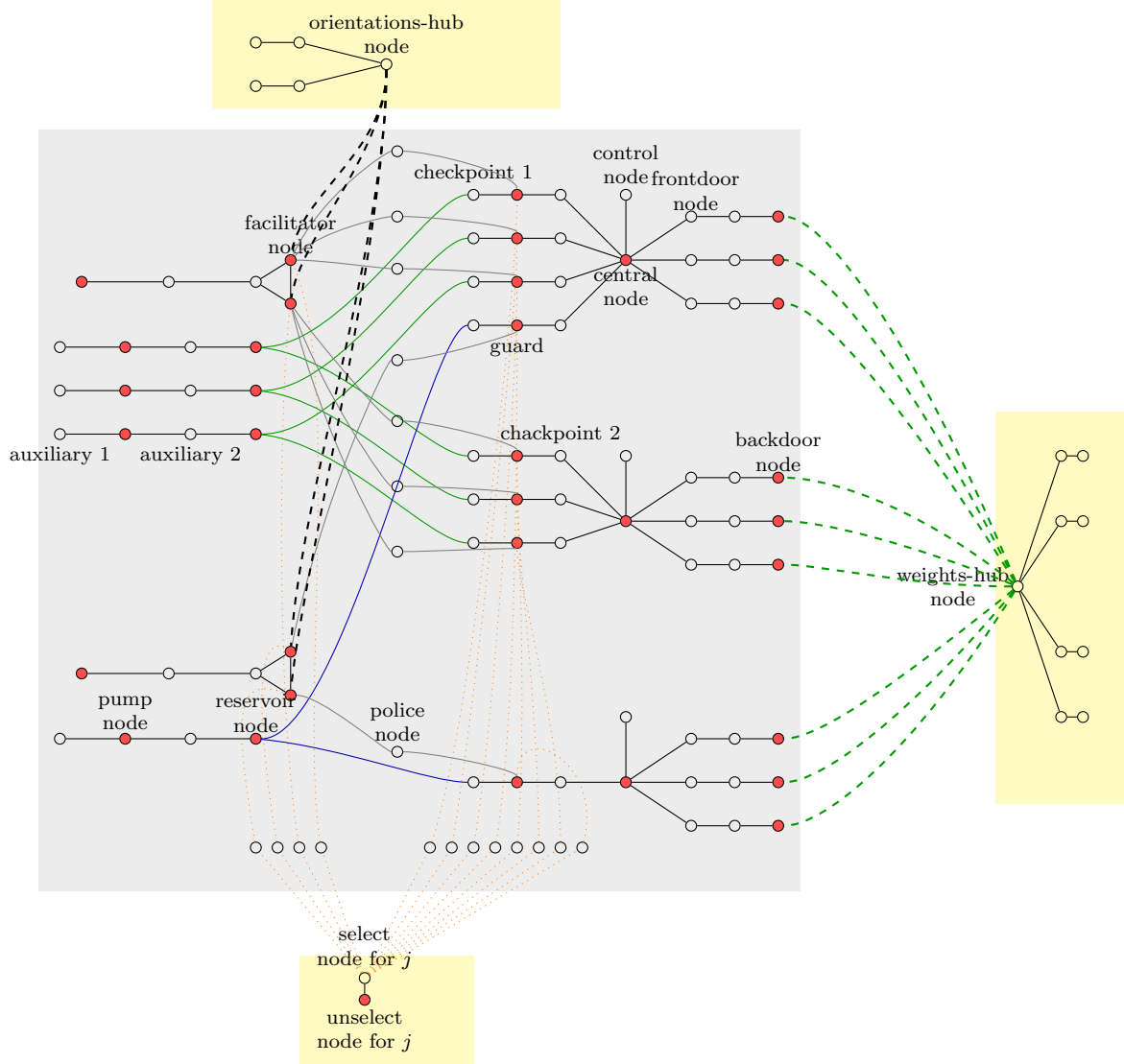


Figure 6.9: Part of the instance selector of the graph H^+ of DOMINATING SET-DISCOVERY (yellow-shaded area) and its connections to one subgraph H^j for $j \in [t]$ (gray-shaded area). Red nodes contain tokens. The orientations-hub node is adjacent to all facilitator nodes (black dashed edges), the weights-hub node is adjacent to all backdoor nodes (green dashed edges), and the select node for j is adjacent to all facilitator and guard nodes in H^j through paths of length two (orange dotted edges).

- each control path becomes a single *control node*,
- each reservoir path becomes a reservoir path of four nodes: the *auxiliary 1 node*, the *pump node*, the *auxiliary 2 node*, and the reservoir node,
- each relay path has two edges connecting checkpoint 1 node to the guard, and the guard to checkpoint 2 node,
- each constraint path has two edges connecting three nodes, and
- each communication path between a guard (v, e) and its corresponding facilitator node (e, v) has two edges with one *police node* in the middle.

Additionally, we make the orientations-hub node adjacent to every facilitator node and the weights-hub node adjacent to every backdoor node in every subgraph H^j for $j \in [t]$. Finally, for each $j \in [t]$, we make the select node for j adjacent to all the facilitator and guard nodes of H^j s. See Figure 6.9.

We place tokens at all the reservoir, pump, central, facilitator, backdoor, guard, and degree-1 constraint path nodes. Additionally, we place a token on unselect for j for each $j \in [t]$.

The high-level idea is that now each weight edge needs at least one token to dominate its vertices, and it will have to take this token from a backdoor node. Additionally, each orientation edge needs at least one token to dominate its vertices, and it will have to take this token from a facilitator node. This will consume $2\Sigma + 2m$ token slides. We tighten the budget so that in addition to these token slides, only $4\Sigma + 1$ token slides are allowed. One token slide can be used to slide the token on the unselect node for j toward the select node for j which then dominates vertices adjacent to the select node for j with no other adjacent tokens. The other 4Σ token slides can be used to fill each frontdoor node on the same receiving path as a now empty backdoor node (Σ of those to slide tokens from reservoir nodes), thus ensuring that we move tokens from distinct backdoor nodes and from the same subgraph H^j . Thus, $b = 2m + 6\Sigma + 1$.

We now show the forward and backward directions of the proof of Theorem 111.

Lemma 112. *If, for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO, then (H^+, S, b) is a yes-instance of DOMINATING SET-DISCOVERY.*

Proof of Lemma 112. Let $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ be a yes-instance of MMO and let $\theta : E(G^j) \rightarrow V(G^j) \times V(G^j)$ be an orientation of G^j such that for each $v \in V(G^j)$, the total weight

of edges directed away from v is at most r^j . In H^+ , for each edge $e \in G^j$ such that $\theta(e) = (v, u)$, we slide the token on facilitator (e, u) into a token-free orientation node. This consumes $2m$ token slides. We slide the tokens on the reservoir nodes in the edge-gadget for e into their adjacent checkpoint 1 nodes in the vertex-gadget for v . This requires Σ token slides.

Now, we slide each token on a guard node adjacent to a checkpoint 1 node containing a token to a token-free frontdoor node in the same gadget. Since this is a feasible orientation, each vertex-gadget receives at most r tokens on its checkpoint 1 nodes. This requires 3Σ token slides.

Since the number of tokens now on distinct frontdoor nodes is Σ , we slide each token on a backdoor node of a receiving path with a frontdoor node containing a token to a distinct token-free weight node. This requires 2Σ token slides. Finally, we slide the token on the unselect node for j into the select node for j using one token slide. Performing these $b = 2m + 6\Sigma + 1$ token slides results in a dominating set in H^+ . $\square$

Before showing the backward direction, we note the following two observations.

Observation 113. *In a solution to an instance (H^+, S, b) of DOMINATING-SET-DISCOVERY that requires the minimum number of token slides, it is safe to assume that the tokens on the central nodes do not slide.*

Observation 114. *In a solution to an instance (H^+, S, b) of DOMINATING-SET-DISCOVERY that requires the minimum number of token slides, no token can slide from a subgraph H^j for some $j \in [t]$ into a weight or orientation node without sliding a token to the select node for j .*

Lemma 115. *If (H^+, S, b) is a yes-instance of DOMINATING SET-DISCOVERY, then for some $j \in [t]$, $(G^j, \mathcal{P}_{G^j}, \sigma^j, r^j)$ is a yes-instance of MMO.*

Proof of Lemma 115. First, Σ tokens have to slide from backdoor nodes into weight nodes. Additionally, m tokens have to slide from facilitator nodes to orientation nodes. This requires $2m + 2\Sigma$ token slides. Furthermore, from Observation 114, we also conclude that at least one token slide is needed to slide a token into the select node for j for some $j \in [t]$. Note that we can safely assume that for some solution that uses the minimum number of token slides, this token comes from the unselect node for j .

This implies that at most 4Σ token slides are left. From Observation 113, each token that slides from a backdoor node of a certain receiving path requires at least three token slides to slide a token into a frontdoor node of the same path from a guard node. Also, when

a token moves from a guard node, it requires at least one token slide to slide a token into either the same node or an adjacent node. Note that this consumes the entire budget, thus when a token slides from a guard node it can force exactly one token slide. Additionally, given the tightness of the budget, it must be the case that tokens move into orientation and weight nodes from the same subgraph H^j for some $j \in [t]$. Moreover, it must be the case that these tokens come from distinct facilitator nodes in m distinct edge-gadgets of H^j into the orientation nodes. Otherwise, we would need at least one additional token slide to dominate all the tokens of a constraint path or to get a token into a facilitator node.

Note that every constraint gadget in H_j can be missing a token on exactly one of its facilitator nodes. When a token moves from a guard node, it must be the case that a token slides from a reservoir node into the checkpoint 1 node adjacent to the guard node. Otherwise, we would require at least one additional token slide. Since Σ tokens slide into the weight nodes from backdoor nodes, and consequently Σ tokens slide into frontdoor nodes, Σ tokens slide from the reservoir nodes. Thus, all the tokens on the reservoir nodes of every edge-gadget slide.

For any edge $e = uv \in E(G^j)$, only if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration, then a guard node of relay paths from v to e (resp. from u to e) cannot contain a token in the final configuration. Thus, so that Σ tokens slide from reservoir nodes, all tokens on the reservoir nodes in the edge-gadget for e slide to the checkpoint 1 nodes of the vertex-gadget for v (resp. u). However, since each vertex-gadget should receive at most r such tokens and consequently send at most r tokens to the weight edges, we can produce a feasible orientation θ of G^j as follows. We set $\theta(e) = (v, u)$ (resp. $\theta(e) = (u, v)$) if the facilitator node (e, v) (resp. (e, u)) contains a token in the final configuration. Since each vertex-gadget for v can send at most r tokens into weight nodes, and thus receive at most r tokens into its checkpoint 1 nodes, θ is such that the total weight directed out of any vertex $v \in V(G)$ is at most r . $\square$

6.3 Polynomial Kernel for Independent Set-Discovery with Parameter k on every nowhere dense class of graphs

In this section, we show that INDEPENDENT SET-DISCOVERY has a polynomial kernel when the parameter is k on every [class of nowhere dense graphs](#) (refer to Definition 2).

First, recall from Section 3.4 of the preliminaries that, for an instance of a solution discovery problem with a given budget b , we call a vertex $u \in V(G)$ *irrelevant with respect to a vertex $v \in S$* , if given a solution $(T, \vec{T})$ where the token on v slides and stays on $u \in T$, we can construct a solution that requires the same number of token slides and where u is not in the final configuration. Note that in the latter definition it is possible for the vertex u to be used for traversing. Additionally, recall from Section 3.1 of the preliminaries, for vertices $u, v \in V(G)$, we denote by $d(u, v)$ the length of a shortest path between u and v , equivalently, the minimum number of token slides needed to get a token on u to slide to v . For a vertex $u \in V(G)$, we set $V(u, i) = \{v \in V(G) \mid d(u, v) = i\}$.

Before presenting the algorithm, we argue that as in the MATCHING-DISCOVERY kernelization with respect to parameter k (Section 5.1.3), there is always a solution that slides any token at most $3k$ token slides.

Claim 116. *Let (G, S, b) be an instance of INDEPENDENT SET-DISCOVERY and let G' be the subgraph of G induced by the vertices of $\bigcup_{u \in S, i \in [3k]} V(u, i) \cup S$. Then (G', S, b) is a yes-instance of INDEPENDENT SET-DISCOVERY if and only if (G, S, b) is.*

Proof of Claim 116. We show that in any solution $(T, \vec{T})$ to (G, S, b) , if a token on a vertex $u \in S$ slides to a vertex $v \in T$ such that $d(u, v) > 3k$, it can instead slide towards a vertex $v' \in V(G)$ such that $d(u, v') \leq 3k$, while keeping the rest of the solution unchanged. First, the vertices in $T \setminus \{v\}$ can appear in at most $k - 1$ of the $3k$ sets $V(u, i)$ for $i \in [3k]$ and every such vertex that appears in a set $V(u, i)$ can be a neighbor of only the vertices in $V(u, i - 1)$ and $V(u, i + 1)$. This implies that the token on u cannot slide toward any vertex of at most $3k - 3$ of the $3k$ sets $V(u, i)$ for $i \in [3k]$ (as these could result in adjacent tokens) and thus there exists a vertex v' to which the token on u can slide while maintaining an independent set in $T \setminus \{v\} \cup \{v'\}$. $\square$

Next, we show that for each vertex $u \in S$, we can decrease the size of each large set $V(u, i)$ for $i \in [3k]$ to a function of k by removing irrelevant vertices with respect to u .

Claim 117. *Let (G, S, b) be an instance of INDEPENDENT SET-DISCOVERY and let $\{v_1, v_2, \dots, v_t\}$ be a set of vertices of $G \setminus S$ such that for a given vertex $u \in S$, $d(u, v_i) = d(u, v_j)$ for $i \neq j \in [t]$. If $\mathcal{A} = \{N[v_1], \dots, N[v_t]\}$ contains a sunflower with $k + 1$ sets, then any vertex whose closed neighborhood corresponds to one of those sets is irrelevant with respect to u .*

Proof of Claim 117. Let v_{del} be one such vertex whose closed neighborhood corresponds to one of the $k + 1$ sets, and consider a solution $(T, \vec{T})$ to (G, S, b) in which the token on u is

on the vertex v_{del} in T . First, note that no vertex of $T \setminus \{v_{del}\}$ is in the core of the sunflower since that would contradict the fact that $v_{del} \in T$ as v_{del} is a neighbor of every vertex in the core. Second, since the sunflower has k remaining sets besides the one corresponding to the closed neighborhood of v_{del} and $|T \setminus \{v_{del}\}| = k - 1$, there must remain one vertex (denoted v_{rep}) whose closed neighborhood corresponds to one of the remaining k sets and such that v_{rep} is not in the closed neighborhood of any of the vertices in $T \setminus \{v_{del}\}$. If it was in the closed neighborhood of a vertex $v'' \in T \setminus \{v_{del}\}$, then $v'' \in N[v_{rep}]$. Thus, $T \setminus \{v_{del}\} \cup \{v_{rep}\}$ forms an independent set. Additionally, since $d(u, v_{del}) = d(u, v_{rep})$, the number of slides required in the new transformation sequence remains the same as the one required for $\vec{T}$ (the transformation sequence corresponding to T). As a result, v_{del} is irrelevant with respect to u . $\square$

To present the kernelization, we will finally need the following result for nowhere dense classes of graphs, which will allow us to create the sunflower needed to apply Claim 117.

Theorem 118 ([133, 159]). *Let $\mathcal{C}$ be a nowhere dense class of graphs. There is a polynomial $N_2 : \mathbb{N} \rightarrow \mathbb{N}$ and a constant $x_2 \in \mathbb{N}$ such that following holds. Let $G \in \mathcal{C}$ and let $A \subseteq V(G)$ be a vertex subset of size at least $N_2(m)$, for a given $m \in \mathbb{N}$. Then there exists a set $X \subseteq V(G)$ of size $|X| \leq x_2$ and a set $B \subseteq A \setminus X$ of size at least m such that no pair of vertices in B have a common neighbor in $G - X$. Moreover, given G and A , such sets X and B can be built in time that is polynomial in n (the size of G).*

Theorem 119. *Let $\mathcal{C}$ be a class of nowhere dense graphs. INDEPENDENT SET-DISCOVERY parameterized by k admits a polynomial kernel.*

Proof. Let (G, S, b) be an instance of INDEPENDENT SET-DISCOVERY where $G \in \mathcal{C}$. Without loss of generality, we assume the graph G to be connected. For each vertex $u \in S$ and integer $i \in [3k]$, we compute $V(u, i)$. We maintain the invariant that we remove from $V(u, i)$, for each $u \in S$ and $i \in [3k]$, irrelevant vertices with respect to u .

We remove an irrelevant vertex with respect to a vertex $u \in S$ from $V(u, i)$ for an integer $i \in [3k]$ as follows. If $|V(u, i)| > N_2(2^{x_2} \cdot (k + 1))$, where N_2 and x_2 are as in Theorem 118 (here $V(u, i)$ plays the role of the set A), we can compute sets $X, B \subseteq V(u, i)$ such that $|X| \leq x_2$, $|B| \geq 2^{x_2} \cdot (k + 1)$ and no pair of vertices in B have a common neighbor in $G - X$. Let $\mathcal{B}' = \{B'_1, B'_2, \dots\}$ be a family of sets that partitions the vertices in B such that for any two vertices $u, v \in B$, $u, v \in B'_j$ if and only if $N[u] \cap X = N[v] \cap X$ (i.e., u and v have the same closed neighborhood in X). Since $|B| \geq 2^{x_2} \cdot (k + 1)$ and $|X| \leq x_2$, at least one set $B_j \in \mathcal{B}$ contains at least $k + 1$ vertices of B . All vertices in B_j have the same neighborhood in X and no vertex from outside of X can be in the closed neighborhood of

two vertices in B_j ; thus their closed neighborhoods form a sunflower with at least $k + 1$ sets and a core that is contained in X (Figure 6.10). By Claim 117, one vertex of B_j is irrelevant with respect to u and can be removed from $V(u, i)$. We can repeatedly apply Theorem 118 on the set $V(u, i)$ until $|V(u, i)| \leq N_2(2^{x_2} \cdot (k + 1))$.

We form the kernel (G', S, b') of the original instance (G, S, b) as follows. Given the above and Claim 116, we set $V(G') = \bigcup_{u \in S, i \in [3k]} V(u, i) \cup S$, and for each vertex $u' \in V(u, i)$, for $u \in S$ and $i \in [3k]$, we add to $V(G')$ at most i vertices that are on the shortest path from u to u' , if such vertices are not already present in $V(G')$. G' is the subgraph of G induced by the vertices in $V(G')$. We set b' to the minimum of b and $k \cdot |V(G')|$.

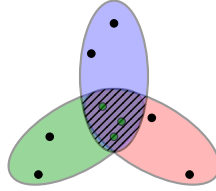


Figure 6.10: An example of a sunflower (with pink, blue and green petals) and a common core (the shaded intersection area). This sunflower is formed by the closed neighborhoods of the vertices in B_j of Theorem 119. This holds because no vertex from outside of X can be in the closed neighborhood of two vertices in B_j and the vertices of B_j have the same closed neighborhood in X (the green colored vertices).

By the end of this process, $|V(G')| \leq k + [3k^2 \cdot N_2(2^{x_2} \cdot (k + 1))] \cdot (1 + (3k - 1))$, as for each $u \in S$ and $i \in [3k]$, $|V(u, i)| \leq N_2(2^{x_2} \cdot (k + 1))$ and for each vertex in those sets, we added to $V(G')$ at most $3k - 1$ vertices that are on a shortest path from that vertex to the vertex u . Additionally, constructing the new instance requires only time polynomial in n . (G', S, b') is a kernel as only vertices that are irrelevant with respect to every token in S might not be in $V(G')$ and all vertices needed to move tokens from vertices in S towards an independent set using at most b token slides are present in $V(G')$. Also, by Claim 10, the budget we need in G' is at most b' . $\square$

6.4 Polynomial Kernel for Dominating Set-Discovery with Parameter k on every biclique-free class of graphs

In this section, we show that DOMINATING SET-DISCOVERY admits a polynomial kernel when the parameter is k on every [class of biclique-free graphs](#) (refer to Definition 1).

For the proof, we use the concept of k -domination cores. For $k \geq 1$, a set $C \subseteq V(G)$ is a *k -domination core* if every set of size at most k that dominates C also dominates G . For our kernelization, it is important that, given a class $\mathcal{C}$ of biclique-free graphs, one can decide whether a biclique-free graph $G \in \mathcal{C}$ has a k -domination core and if so, compute one of size that is polynomial in k , in polynomial time. The following theorem states that this is possible. Note that the notion of cores does not appear explicitly in [158], however, it is easily observed that the set of black vertices in the auxiliary RWB-dominating set problem considered in that work yields a k -domination core.

Lemma 120 (follows from [158]). *Let $\mathcal{C}$ be a class of biclique-free graphs. There exists a polynomial-time algorithm that, given $G \in \mathcal{C}$ and $k \in \mathbb{N}$ decides that $V(G)$ cannot be dominated by k vertices or computes a k -domination core $C \subseteq V(G)$ of polynomial size in k .*

Additionally, for the proof, we require that given a class of biclique-free graphs, a graph G in that class, and a k -domination core C for G , the number of distinct neighborhoods of vertices in $V(G) \setminus C$ in C is polynomially bounded in $|C|$. This holds, as stated in Corollary 123. To show the corollary, we need the concept of the *VC-dimension* of a graph. The VC-dimension of a graph G is the maximum size of a vertex subset $S \subseteq V(G)$ that can be *shattered*, which means that for every subset $S' \subseteq S$, there exists a vertex $v \in V(G)$ such that $N(v) \cap S = S'$. In a d -biclique-free graph, the VC-dimension is at most $2d$, as stated in the following theorem. The theorem is trivial since if S is a set of size at least $2d$ that can be shattered in G , then it suffices to look for the d vertices in $V(G)$ that have in their neighborhood in S the same d vertices and only one additional distinct vertex for d distinct vertices. By definition, we can find these d vertices that are all neighbors of the same d vertices in S , and thus G cannot be d -biclique-free.

Theorem 121 ([167]). *If a graph G is d -biclique-free, then the VC-dimension of G is at most $2d$.*

The bounded VC-dimension of biclique-free graphs combined with the following theorem leads to Corollary 123.

Theorem 122 (Sauer-Shelah Lemma [164, 166]). *Let $\mathcal{F}$ be a set system on a ground set X with VC-dimension at most d . Then $|\mathcal{F}| \leq \sum_{i=0}^d \binom{|X|}{i} \leq c_d \cdot |X|^d$ where c_d is a constant that depends on d .*

Corollary 123. *Let G be a d -biclique-free graph, and $C \subseteq V(G)$. The number of distinct sets in $\{N(v) \cap C : v \in V(G)\}$ is at most $c_d \cdot |C|^d$ where c_d is a constant that depends on d .*

This upper bound on the number of distinct neighborhood patterns, as opposed to the $2^{|C|}$ possible in general graphs, will be key to the efficiency of the algorithm presented in the proof of Theorem 124.

Theorem 124. *Let $\mathcal{C}$ be a class of biclique-free graphs. DOMINATING SET-DISCOVERY parameterized by k admits a polynomial kernel.*

Proof of Theorem 124. Let (G, S, b) be an instance of DOMINATING SET-DISCOVERY, where $G \in \mathcal{C}$. We first compute a k -domination core $C \subseteq V(G)$ of polynomial size in k , which is possible by Lemma 120. We then partition in polynomial time the vertices in $V(G) \setminus C$ into equivalence classes where two vertices $u, v \in V(G) \setminus C$ are equivalent if and only if $N(u) \cap C = N(v) \cap C$. By Corollary 123, the number of such equivalence classes is polynomially bounded in $|C|$. For a set $M \subseteq V(G)$ and a vertex $v \in V(G)$ we let $d(v, M) = \min_{w \in M} d(v, w)$. For every equivalence class X we now fix a minimal set MX of *representative vertices* such that for each vertex $v \in S$ the set MX contains a vertex u such that $d(v, u) = d(v, X)$. Note that MX contains at most k vertices and that such a set can be computed in polynomial time by a simple [breadth-first search](#). We define $U \subseteq V(G)$ as the union of the vertices of C , the vertices of S , and the vertices of a shortest path between v and its corresponding representative vertex u for each $v \in S$ and equivalence class X . We let G' be the graph induced by the vertices in U . We define the kernel as (G', S, b') where we set b' to be the minimum of b and $k \cdot |V(G')|$. Clearly, constructing the new instance requires only time polynomial in n .

First we prove that (G', S, b') is a valid kernel. First assume that (G, S, b) is a yes-instance and let $(T, \vec{T})$ be a solution that uses the minimum number of token slides. As T is a dominating set, it dominates in particular the core C . Assume that a token t on a vertex $v \in S$ is destined for a vertex in the equivalence class X . Then we obtain a solution by making the token t destined for the representative vertex u corresponding to v and the equivalence class X instead while maintaining the destinations of all other tokens. This transformation sequence exists in G' and ends in a set of size at most k that dominates C . Hence, it also dominates G' . Additionally, by Claim 10, the budget we need in G' is at most b' . This shows that (G', S, b') is a yes-instance. Conversely, a transformation sequence

in G' to a dominating set of G' leads to a dominating set of C in G' . By definition of a k -domination core, we also discover a dominating set in G by applying the transformation sequence of G' which has at most b token slides.

Finally, it remains to show that (G', S, b') has size bounded by a polynomial in k . As we argued already, we have a polynomial-size core C and at most polynomially many equivalence classes. From each class, we keep at most k representative vertices and at most some polynomial in k vertices of shortest paths. Additionally, this implies that b' is a function of k . $\square$

Chapter 7

Investigating Meta-Theorems for Solution Discovery

In this chapter, we move beyond analyzing individual problems to understand entire families of solution discovery problems at once through the lens of descriptive complexity. Rather than repeatedly proving similar results for each graph property (such as analyzing the complexity of each of INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY with respect to parameter pathwidth), we investigate theorems that simultaneously capture the tractability of all solution discovery problems definable in certain logical languages. In particular, we work towards answering “When the final configuration must satisfy a property expressible in MSO or FO, which structural graph parameters enable efficient algorithms?”

On the positive side, we show in Section 7.1 that MSO-DISCOVERY is fixed-parameter tractable when parameterized by neighborhood diversity. This positive result generalizes many problem-specific results (such as INDEPENDENT SET-DISCOVERY, VERTEX CUT-DISCOVERY, and others).

On the other hand, in Sections 7.2 and 7.3, we show that even FO-DISCOVERY, despite its weaker expressiveness, is $W[1]$ -hard when parameterized by modulator to stars, modulator to paths and twin cover numbers. We note that FO-DISCOVERY includes all of VERTEX COVER-DISCOVERY, INDEPENDENT SET-DISCOVERY, DOMINATING SET-DISCOVERY, and SHORTEST PATH-DISCOVERY, but not VERTEX CUT-DISCOVERY, which is included in MSO-DISCOVERY since quantification over a set is necessary to represent the property of a vertex cut. Additionally, we note that showing that FO-DISCOVERY (MSO-DISCOVERY, respectively) is $W[1]$ -hard means that there exists at least one prop-

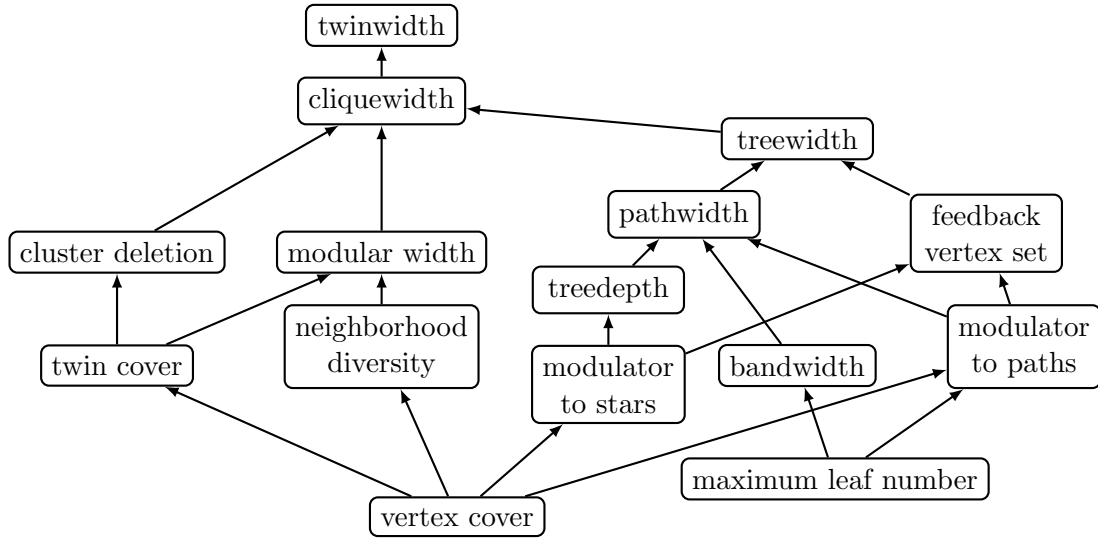


Figure 7.1: A subset of possible structural parameters. An arrow between two boxes indicates the existence of a function in the parameter above that lower-bounds the parameter below.

erty Π definable in FO (MSO, respectively) for which Π -DISCOVERY is $W[1]$ -hard. The $W[1]$ -hardness results of Sections 7.2 and 7.3 demonstrate that the dynamic nature of solution discovery fundamentally changes the computational landscape compared to static MODEL CHECKING. This is true because for parameters like modulator to stars, paths and twin cover numbers where MODEL CHECKING can be efficient, FO-DISCOVERY is $W[1]$ -hard.

In Section 7.4, we show that MSO-DISCOVERY is $W[1]$ -hard when parameterized by bandwidth through a reduction from the PLANAR ARC SUPPLY problem (which we define in the same section). With these findings, we cover most of the structural parameters in Figure 2.1 (reproduced in Figure 7.1 for convenience). Note that prior to the results presented in this section, the only known results with respect to any of the parameters in Figure 7.1 are that VERTEX COVER-DISCOVERY, INDEPENDENT SET-DISCOVERY, and DOMINATING SET-DISCOVERY are $XNLP$ -hard when parameterized by pathwidth (these results were presented in Section 6.1), thus we cannot expect to obtain a meta fixed-parameter tractability theorem for this parameter.

Finally, we remind the reader that when we work with FO and MSO, we consider vertex-colored graphs where each vertex can have more than one color.

The results of this chapter appear in the manuscript titled “*Surveying Graph Paramete-*

ters for Algorithmic Meta-Theorems for Solution Discovery” and co-authored with Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz.

7.1 Fixed-Parameter Tractability of MSO-Discovery for Neighborhood Diversity

In this section, we show that MSO-DISCOVERY is fixed-parameter tractable when parameterized by the neighborhood diversity.

Recall from Section 3.6 of the preliminaries that we say that two vertices $u, v \in V(G)$ are *twins* if $N(u) = N(v)$ or $N[u] = N[v]$. The *neighborhood diversity* of a graph G , denoted by $nd(G)$, is the number of subsets in the unique partition of $V(G)$ into maximal sets of twin vertices called twin classes. It is known that the neighborhood diversity and the corresponding partition of a graph can be computed in linear time [144, 172].

For a vertex-colored graph G , let $V_1, \dots, V_\ell$ be the partition of $V(G)$ into sets of vertices such that any two vertices u and v in V_i for $i \in [\ell]$ are twins and have the same colors (that is, $\mathcal{C}(u) = \mathcal{C}(v)$). We refer to each V_i as a *vertex-type* and say that two vertices $u, v \in V_i$ are of the *same vertex-type*. This partition can be computed in polynomial time and the number ℓ depends only on $nd(G)$ and on the number of colors in $\mathcal{C}$.

For this fixed-parameter tractable algorithm, we will require the concept of *shapes* of vertex subsets [129]. The key insight is that, when dealing with logical formulas, many different vertex subsets behave identically from the formula’s perspective. Intuitively, the shape records how many vertices a vertex subset contains from each vertex-type, but with a crucial optimization. When the subset R contains a “medium” number of vertices from a vertex-type V_i (not too few, not too many relative to some number), the exact count becomes irrelevant for the truth value of the formula. The shape abstracts this by marking such counts as $\perp$.

Definition 125 (Shape). *Let $\phi(X)$ be an MSO-formula with a free set variable X , and let q_s and q_v be the numbers of set and vertex quantifiers in ϕ , respectively. Let G be a graph with ℓ vertex-types $V_1, \dots, V_\ell$, and define $q(\phi) = 2^{q_s} \cdot q_v$. Let $R \subseteq V(G)$ and let $\sigma_R(i) = |R \cap V_i|$. The shape of R is defined as follows:*

$$\bar{\sigma}_R(i) = \begin{cases} \perp & \text{if } q(\phi) \leq \sigma_R(i) \leq |V_i| - q(\phi), \\ \sigma_R(i) & \text{otherwise,} \end{cases}$$

The following result of Gimma et al. [99] is central to our algorithm.

Lemma 126. *If two size- k sets $R, R' \subseteq V(G)$ have the same shape, then $G \models \phi(R)$ if and only if $G \models \phi(R')$.*

This abstraction dramatically reduces the search space when searching for a set that satisfies the formula. Instead of considering all $\binom{n}{k}$ possible vertex subsets of size k , we only need to examine one subset of each distinct shape, and the number of shapes depends on the formula size and the graph structure, rather than the graph size. This technique is particularly powerful when combined with neighborhood diversity as a parameter, leading to a bounded (with respect to the parameter) number of possible shapes.

Additionally, we will require a fixed-parameter tractable algorithm (with respect to neighborhood diversity and $|\phi|$) for the MINIMUM-COST FLOW WITH INTERVAL DEMAND AND SUPPLY problem (MINMCF) and a fixed-parameter tractable algorithm (with respect to neighborhood diversity and $|\phi|$) for checking, given a formula $\phi(X)$ and a set $S \subseteq V(G)$ whether $G \models \phi(S)$. The latter exists for parameter neighborhood diversity and $|\phi|$ by a corollary from Lampis [136].

MINMCF is solvable in polynomial time [119] and is defined on a digraph D as follows. Each arc a in D has a maximum capacity $c(a)$ and a unit of flow cost $p(a)$. Each vertex v in D is assigned an interval $[I_v^{\min}, I_v^{\max}]$. A flow is the assignment of a value $f(a)$ to each arc such that:

- $0 \leq f(a) \leq c(a)$ for all $a \in A(D)$, and
- the *net balance* at each vertex $v \in V(D)$, denoted by $b(v) := \sum_{u:(u,v) \in A} f(u,v) - \sum_{w:(v,w) \in A} f(v,w)$, is in the interval $[I_v^{\min}, I_v^{\max}]$ (a positive net balance creates a sink, a negative net balance creates a source).

The objective of the problem is to answer whether there exists a flow with a total cost $\sum_{a \in A(D)} p(a)f(a)$ at most ℓ .

We are now ready to present the fixed-parameter tractable algorithm.

Theorem 127. *MSO-DISCOVERY is in FPT when parameterized by neighborhood diversity.*

Proof of Theorem 127. Let $((G, \mathcal{C}), S, b, \phi)$ be an instance of MSO-DISCOVERY where $\mathcal{C}$ is the set of colors, S is the initial set, b is the budget, and ϕ is the formula (which is assumed to be of constant size or part of the parameter). Let ℓ be the number of vertex-types in $(G, \mathcal{C})$; then $\ell \leq nd(G) \cdot 2^{|\mathcal{C}|} \leq nd(G) \cdot 2^{|\phi|}$. We enumerate every admissible shape $\bar{\sigma}_T : [\ell] \rightarrow \{0, \dots, q(\phi) - 1, \perp, |V_i| - q(\phi) + 1, |V_i| - q(\phi) + 2, \dots, |V_i|\}$ of the final configuration T (of size k). There are only ℓ vertex-types and $q(\phi)$ is a parameter, so

this yields a bounded number $((2q(\phi) + 1)^\ell)$ of shapes, which we call *guesses*. Using a flow argument, we will now prove that, for each guess, we can compute in fixed-parameter tractable (in fact polynomial) time the minimum budget needed for a configuration of that shape (if any exists).

Let $\bar{\sigma}_T$ be a guess. Let us first give an algorithm simulating the token sliding process using a flow in a digraph. For each vertex-type V_i of $(G, \mathcal{C})$, we create a vertex v_i in the digraph D and connect the vertices v_i, v_j (in both directions) if and only if all vertices in V_i are adjacent to all vertices in V_j (recall that vertices of the same type are twins in the graph). Arcs of D have infinite capacity and a unit of flow cost of one. Each vertex v_i in D has:

- an interval equal to $[-(|V_i \cap S| - \sigma_T(i)), -(|V_i \cap S| - \sigma_T(i))]$ if $\bar{\sigma}_T(i) \neq \perp$ and $|V_i \cap S| > \sigma_T(i)$;
- an interval equal to $[\sigma_T(i) - |V_i \cap S|, \sigma_T(i) - |V_i \cap S|]$ if $\bar{\sigma}_T(i) \neq \perp$ and $|V_i \cap S| < \sigma_T(i)$;
- an interval equal to $[q(\phi) - |V_i \cap S|, |V_i| - q(\phi) - |V_i \cap S|]$ if $\bar{\sigma}_T(i) = \perp$ and $|V_i \cap S| < q(\phi)$;
- an interval equal to $[-(|V_i \cap S| - |V_i| + q(\phi)), (|V_i \cap S| - q(\phi))]$ if $\bar{\sigma}_T(i) = \perp$ and $|V_i \cap S| > |V_i| - q(\phi)$; and
- an interval equal to $[0, 0]$, otherwise.

We now ask whether there exists a flow in D with a total cost of at most b . This can be done in fixed-parameter tractable time (using the algorithm for MINMCF [119]) since the number of vertex-types, thus vertices in D , is bounded. Then, for each guess $\bar{\sigma}_T$ with a flow of total cost at most b , we construct a k -size set of vertices $T \subseteq V(G)$ of shape $\bar{\sigma}_T$ in linear time. We check whether $G \models \phi(T)$, which can be done in fixed-parameter tractable time with respect to $nd(G)$ and $|\phi|$, and return YES only when $G \models \phi(T)$. If we have exhausted all enumerated shapes and did not return YES, we return NO.

Since all steps involve a bounded number of guesses (equal to $(2q(\phi) + 1)^\ell$ with $\ell \leq nd(G) \cdot 2^{|\mathcal{C}|} \leq nd(G) \cdot 2^{|\phi|}$ since we are forming every possible mapping of $[\ell]$ to $(2q(\phi) + 1)$ numbers), and for each guess we incur fixed-parameter tractable running time with respect to $nd(G)$ and $|\phi|$ (in particular $2^{\mathcal{O}(2^{|\phi|} \cdot nd(G) \cdot |\phi| + |\phi| \log |\phi|)}$), the overall algorithm is fixed-parameter tractable with respect to $(nd(G), |\phi|)$.

The algorithm is correct since, if for a fixed MSO-formula ϕ , a set S , and a budget b , a final configuration T^* exists along with a transformation sequence $\vec{T}^*$ of at most b token slides that realizes T^* from S , then for at least one guess T we will have $\bar{\sigma}_T = \bar{\sigma}_{T^*}$.

Given that the sequence $\vec{T}^*$ encounters at most b token slides between vertex-types, and achieves the set T^* of shape $\bar{\sigma}_{T^*}$, routing one unit of flow for every inter-vertex-type slide in $\vec{T}^*$ yields a flow of cost at most b . In the final checking phase, the algorithm will be able to construct a set of shape $\bar{\sigma}_T = \bar{\sigma}_{T^*}$, and since T^* is a valid final configuration, the set constructed by the algorithm will also be a valid final configuration by Lemma 126 and therefore, the algorithm will return YES.

Conversely, if the algorithm accepts, then there exists a flow that satisfies the interval and cost constraints imposed by some guessed fixed shape $\bar{\sigma}_T$ and the budget b . A unit of flow sent along an arc in D represents sliding one token across an edge of G that joins the two corresponding vertex-types; since each arc in D between vertices representing vertex-types has a unit of flow cost of one, a flow of total cost at most b encodes b token slides between vertex-types. The intervals guarantee that, after performing those slides imposed by the flow, every type V_i contains exactly $\sigma_T(i)$ tokens if $\bar{\sigma}_T(i) \neq \perp$ and between $q(\phi)$ and $|V_i| - q(\phi)$ tokens otherwise. Thus, the set T^* , resulting from performing those slides imposed by the flow on G , has the guessed shape $\bar{\sigma}_T$ and is a valid final configuration as is any other set of the shape $\bar{\sigma}_T$ which has lead to acceptance (Lemma 126). $\square$

7.2 Hardness of FO-Discovery for Modulator to Stars and Modulator to Paths Numbers

In this section, we show that FO-DISCOVERY is W[1]-hard when parameterized by the parameters modulator to stars and modulator to paths numbers.

Recall from Section 3.6 of the preliminaries that a *modulator to paths* is a set of vertices $X \subseteq V(G)$ such that $G - X$ is a disjoint union of paths. The size of a minimum such modulator, denoted by $mp(G)$, is called the *modulator to paths number*. Similarly, a *modulator to stars* is a set of vertices $X \subseteq V(G)$ such that $G - X$ is a disjoint union of stars. The size of a minimum such modulator, denoted by $ms(G)$, is referred to as the *modulator to stars number*.

Before giving the proofs for this section's results, we explain the different gadgets we use in each of the reductions.

For both results, we reduce from the MULTICOLORED CLIQUE problem, which we defined in Section 5.1.4, and we recall here. In the MULTICOLORED CLIQUE problem, we are given a graph G where $V(G)$ is partitioned into κ color classes, which we denote by $\{V_1, \dots, V_\kappa\}$ (to avoid confusion with the color classes in our construction). The goal

is to decide whether there exists a clique that contains a vertex from each color class. The problem is $W[1]$ -hard when parameterized by the solution size κ [58]. Without loss of generality, we will assume, throughout our reductions, that for each $i \in [\kappa]$, $|V_i| = n$ (we can add vertices of degree zero as needed). For each $i \in [\kappa]$, we assign a distinct index in $[n]$ to each vertex in V_i and we refer to this index for each $v \in V$ as $\iota(v)$. In an instance (G, κ) of the MULTICOLORED CLIQUE problem, the edge set $E(G)$ can be partitioned into $\binom{\kappa}{2}$ sets $E_{i,j} = \{uv \mid u \in V_i, v \in V_j, uv \in E(G)\}$ for $1 \leq i < j \leq \kappa$.

Let (G, κ) be an instance of MULTICOLORED CLIQUE. At a high level, token movements in the instance $((H, \mathcal{C}), S, b, \phi)$ of FO-DISCOVERY will encode both the selection of one vertex from each color class and one edge between each unordered pair of color classes, and the budget will force edge verification (that is, that the selected vertices form a clique). For clarity, we refer to *vertices* in MULTICOLORED CLIQUE and *nodes* in our construction.

The construction creates a *vertex-block* for each color class V_i and an *edge-block* for each unordered pair of color classes. A vertex-block for color class V_i , denoted *vertex-block for i* , contains n *vertex-gadgets* (one for each vertex in V_i), plus a special *block node* b_i that connects all vertex-gadgets in the block. Similarly, an edge-block for an unordered pair of color classes V_i and V_j , denoted *edge-block for $\{i, j\}$* , contains *edge-gadgets* (one for each edge between V_i and V_j), plus a special block node $b_{i,j}$. The gadgets themselves have the appropriate structure. That is, they are stars in the modulator to stars reduction, and paths in the modulator to paths reduction, which ensures that they are not included in the modulator. All vertex-blocks are identical, all edge-blocks are identical, all vertex-gadgets are identical, and all edge-gadgets are identical; information is encoded entirely in how gadgets are connected to *connector nodes*. Initially, the leaves of the stars and the paths constituting vertex-gadgets are filled with tokens, and nodes constituting edge-gadgets are empty.

The FO-formula specifies that for each vertex-block, exactly one vertex-gadget releases its tokens (encoding vertex selection) and for each edge-block exactly one edge-gadget, or all its leaves in the case of the modulator to stars reduction, is filled with tokens (encoding edge presence between selected vertices).

The reduction employs connector nodes that act as controlled gateways between vertex-blocks and edge-blocks. For each unordered pair of color classes V_i and V_j , we introduce four *connector nodes for $\{i, j\}$* . Equivalently, we introduce two pairs of connector nodes, *connector nodes from color i to $\{i, j\}$* and *connector nodes from color j to $\{i, j\}$* ;

1. the *index connector node from color i to $\{i, j\}$* which we call for brevity $index(i, \{i, j\})$,
2. the *remainder connector node from color i to $\{i, j\}$* which we call for brevity $remainder(i, \{i, j\})$,

3. the *index connector node* from color j to $\{i, j\}$ which we call for brevity $index(j, \{i, j\})$, and
4. the *remainder connector node* from color j to (i, j) which we call for brevity $remainder(j, \{i, j\})$,

with carefully designed adjacencies. A connector node from color i to $\{i, j\}$ indicates that the vertex-block for i will serve as the origin of the tokens and the edge-block for $\{i, j\}$ will serve as the destination of the tokens that will traverse this connector.

The connections from a vertex-gadget representing v in V_i to the connector nodes for $\{i, j\}$ encode the index of v by having $\iota(v)$ (non-block) nodes of the gadget adjacent only to $index(i, \{i, j\})$ among all connector nodes and $n - \iota(v)$ other (non-block) nodes of the gadget adjacent only to $remainder(i, \{i, j\})$ among all connector nodes. Connections from an edge-gadget representing edge uv between V_i and V_j to the connector nodes for $\{i, j\}$ encode both indices $\iota(u)$ and $\iota(v)$, by having $\iota(u)$ (non-block) nodes of the gadget adjacent only to $index(i, \{i, j\})$ among all connector nodes, $n - \iota(u)$ other (non-block) nodes of the gadget adjacent only to $remainder(i, \{i, j\})$ among all connector nodes, $\iota(v)$ other (non-block) nodes of the gadget adjacent only to $index(j, \{i, j\})$ among all connector nodes, and $n - \iota(v)$ other (non-block) nodes of the gadget adjacent only to $remainder(j, \{i, j\})$ among all connector nodes. This creates a “lock-and-key” mechanism: only the vertex-gadget representing u can send tokens from its nodes connected to the connector nodes from color i to $\{i, j\}$ to the edge-gadget representing edge uv using only two token slides per token, and only the vertex-gadget representing v can send tokens from its nodes connected to the connector nodes from color j to $\{i, j\}$ to the edge-gadget representing edge uv using only two token slides per token. In other words, tokens can only flow efficiently (using two token slides per token, which is imposed by the budget) from any two vertex-gadgets to an edge-gadget (through connectors) if the indices match (that is, an edge exists between these two vertices in G). Thus, a valid token movement pattern exists if and only if the selected vertices form a multicolored clique.

The modulator consists of all the block nodes and connector nodes, which total at most $\kappa + \binom{\kappa}{2} + 4\binom{\kappa}{2}$, and removing it leaves a disjoint union of stars (or paths), ensuring that the parameters are bounded by a function of κ .

We now present in greater detail the gadgets, blocks, and budget. In the subsequent subsections, we give the formula and proofs specific to each parameter.

As mentioned earlier, the construction consists of identical vertex-blocks (made of identical vertex-gadgets) and identical edge-blocks (made of identical edge-gadgets) which are joined together by connector vertices.

To form the colored graph $(H, \mathcal{C})$, we first create κ vertex-blocks, one for each color class V_i in the instance (G, κ) of MULTICOLORED CLIQUE.

Star vertex-block. A *star vertex-block* for i consists of a *vertex block node* b_i that has as children the roots of n stars (that is, star vertex-gadgets), one for each vertex v in V_i . The *star vertex-gadget* for v is a star with a *vertex root node* for v , and $\kappa - 1$ groups of n leaves, where each group corresponds to one of the $\kappa - 1$ color classes that differ from the color class V_i . See Figure 7.2.

We assign the following colors to the nodes in a star vertex-block:

- purple (C_1) to all vertex root nodes, and
- teal (C_2) to all vertex block nodes.

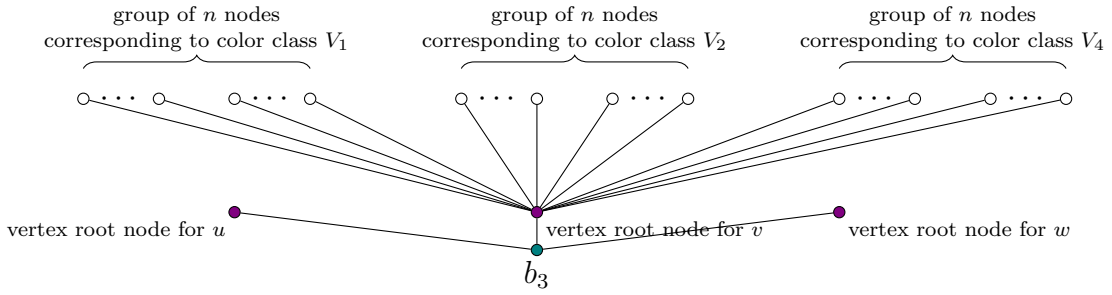


Figure 7.2: A star vertex-gadget for vertex $v \in V_3$ with $\iota(v) = x$, for an instance with $\kappa = 4$, the vertex root nodes for $u, w \in V_3$, and the vertex block node b_3 .

We also create $\binom{\kappa}{2}$ edge-blocks, one for each unordered pair of color classes V_i and V_j .

Star edge-block. A *star edge-block* for $\{i, j\}$ consists of an *edge block node* $b_{i,j}$ that has as children the roots of multiple stars (that is, star edge-gadgets), one for each edge between V_i and V_j . The *star edge-gadget* for e is a star with a *edge root node* for e , and two groups of n leaves, each corresponding to one endpoint of e . See Figure 7.3.

We assign the following colors to the nodes in a star edge-block:

- black (C_3) to all edge root nodes, and
- gray (C_4) to all edge block nodes.

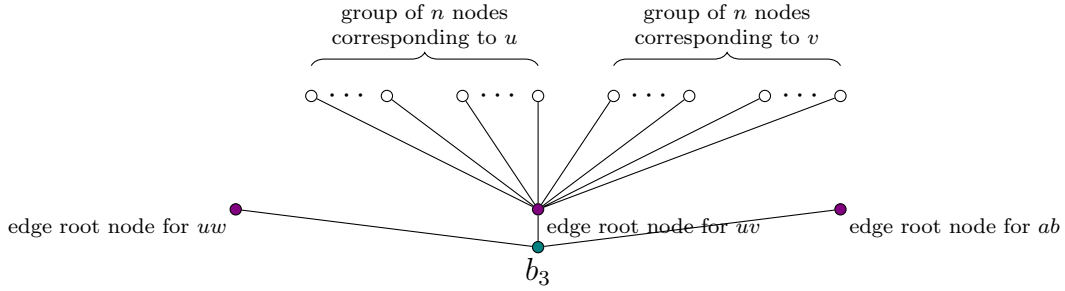


Figure 7.3: A star edge-gadget representing $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$, the edge block nodes for uw and ab (edges between V_i and V_j), and the edge block node b_{ij} . The edge-gadget for uv is formed of $2n$ nodes, n corresponding to u and n corresponding to v .

In the case of the modulator to paths reduction, the vertex-blocks and edge-blocks are constructed as follows.

Path vertex-block. A *path vertex-block* for i consists of a *vertex block node* b_i adjacent to the nodes of n paths (that is, path vertex-gadgets), one for each vertex v in V_i . The *path vertex-gadget* for v is a path with $n(\kappa - 1)$ nodes, divided into $\kappa - 1$ groups of n consecutive nodes, where each group corresponds to one of the $\kappa - 1$ colors that differ from the color class V_i . See Figure 7.4.

We assign teal (C_2) to all vertex block nodes in a path vertex-block.

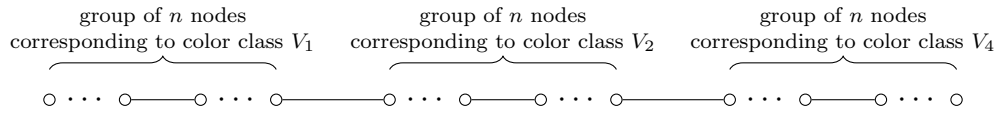


Figure 7.4: A path vertex-gadget for vertex $v \in V_3$ with $\iota(v) = x$, for an instance with $\kappa = 4$. The vertex block node b_3 is adjacent to all nodes of path vertex-gadgets for all vertices in V_3 , but it is omitted from the figure for clarity.

Path edge-block. A *path edge-block* for $\{i, j\}$ consists of an *edge block node* $b_{i,j}$ adjacent to the nodes of multiple paths (that is, path edge-gadgets), one for each edge between V_i and V_j . The *path edge-gadget* for e is a path with $2n$ nodes, with the *first group* of n nodes corresponding to one endpoint of e and the *last group* of n nodes corresponding to the other endpoint. See Figure 7.5.

We assign gray (C_4) to all edge block nodes in a path edge-block.

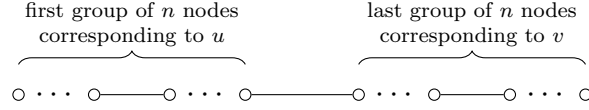


Figure 7.5: A path edge-gadget for edge $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$. It is formed of $2n$ nodes, n corresponding to u and n corresponding to v . The edge block node b_{ij} is adjacent to all nodes of path vertex-gadgets for all edges between V_i and V_j , but it is omitted from the figure for clarity.

Then, for each unordered pair of color classes V_i and V_j , we create four connector nodes; index connector node from color i to $\{i, j\}$ which we call for brevity $\text{index}(i, \{i, j\})$, remainder connector node from color i to $\{i, j\}$ which we call for brevity $\text{remainder}(i, \{i, j\})$, index connector node from color j to $\{i, j\}$ which we call for brevity $\text{index}(j, \{i, j\})$, remainder connector node from color j to $\{i, j\}$ which we call for brevity $\text{remainder}(j, \{i, j\})$. See Figure 7.6. We assign green (C_5) to all connector nodes.

We connect vertex-blocks and edge-blocks to green connector nodes as follows.

- For the vertex-gadget for the vertex $v \in V_i$ with $\iota(v) = x$, the n nodes corresponding to the color class V_j are split as follows: the first x are adjacent to $\text{index}(i, \{i, j\})$, and the last $n - x$ are adjacent to $\text{remainder}(i, \{i, j\})$.
- For the edge-gadget for the edge e with endpoints $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$, the n nodes corresponding to x are split as follows: the first x are adjacent to $\text{index}(i, \{i, j\})$, and the last $n - x$ are adjacent to $\text{remainder}(i, \{i, j\})$. The n nodes corresponding to y are split as follows: the first y are adjacent to $\text{index}(j, \{i, j\})$, and the last $n - y$ are adjacent to $\text{remainder}(j, \{i, j\})$.

As described before, initially, the leaves of the stars and the paths constituting vertex-gadgets are filled with tokens.

So that tokens can travel from vertex-gadgets to fill the nodes of one edge-gadget for each edge-block using exactly two token slides per token (that is, while checking that the indices match between the vertices and the edges), we set $b = 2n(\kappa - 1)\kappa$. We are now ready to give the formula and proofs specific to each parameter.

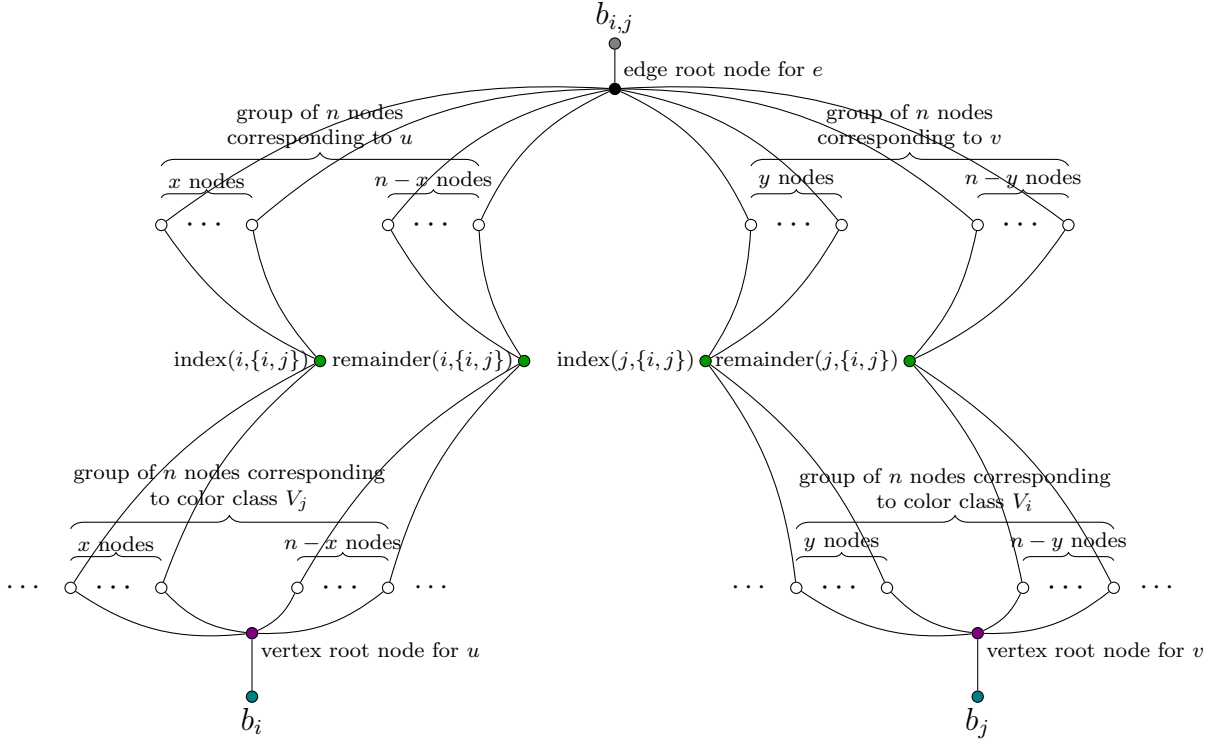


Figure 7.6: Part of the colored graph $(H, \mathcal{C})$ formed by the reduction corresponding to the modulator to stars number. It shows the star edge-gadget of an edge $e = uv$ where $u \in V_i$ and $v \in V_j$ with $\iota(u) = x$ and $\iota(v) = y$, and relevant parts of the two star vertex-gadgets for u and v . Additionally, the block nodes b_i , b_j , and $b_{i,j}$ are shown connecting to their respective root nodes. For u , the first x nodes of the group corresponding to V_j are adjacent to $\text{index}(i, \{i, j\})$ and the remaining $n - x$ nodes are adjacent to $\text{remainder}(i, \{i, j\})$. Similarly, for v , the first y nodes of the group corresponding to V_i are adjacent to $\text{index}(j, \{i, j\})$ and the remaining $n - y$ nodes are adjacent to $\text{remainder}(j, \{i, j\})$.

7.2.1 Hardness of FO-Discovery for Modulator to Stars Number

We start with the parameter modulator to stars number. More formally, the goal of this section is to prove the following:

Theorem 128. *FO-DISCOVERY is $W[1]$ -hard when parameterized by modulator to stars number.*

The rest of this subsection is devoted to the proof of Theorem 128. Let (G, κ) be an

instance of the MULTICOLORED CLIQUE problem. We consider the graph $(H, \mathcal{C})$ described at the start of Section 7.2 where the edge- and vertex-blocks of H are star edge-blocks and star vertex-blocks, the set S consisting of all leaves of all star vertex-gadgets, and the budget $b = 2n(\kappa - 1)\kappa$. We complete $((H, \mathcal{C}), S, b)$ with an FO-formula ϕ to create an instance of FO-DISCOVERY. We form the FO-formula $\phi(X)$ to express that the set X satisfies the following conditions:

S1. No colored node contains a token. (That is, the block nodes, the root nodes, and the connector nodes do not contain tokens.)

S2. For each teal (vertex block) node, there exists exactly one adjacent purple (vertex root) node such that all the uncolored neighbors (leaves) of this purple node do not contain tokens. Moreover, all the other purple (vertex root) nodes adjacent to this teal node have tokens on all the uncolored nodes (leaves) in their neighborhoods. (That is, in a vertex-block, exactly one vertex-gadget has released all the tokens on its leaves, while all other vertex-gadgets retain tokens on all their leaves.)

S3. For each gray (edge block) node, there exists exactly one adjacent black (edge root) node such that all the uncolored neighbors (leaves) of this black node contain tokens. Moreover, all the other black (edge root) nodes adjacent to this gray node are such that all the uncolored nodes (leaves) in their neighborhoods are token-free. (That is, in an edge-block, all but one edge-gadget have no tokens on their leaves and one edge-gadget has tokens on all its leaves.)

The construction of these expressions in FO is presented in Appendix A.1.1. Before diving into the proof, let us reiterate some of the intuition. Each vertex-gadget initially contains $n \cdot (\kappa - 1)$ tokens. This corresponds to n tokens for each of the other color classes that the vertex does not belong to. To transform the current set of tokens into a valid final configuration, we proceed as follows. For every $i \in [\kappa]$, we must empty exactly one vertex-gadget in the vertex-block for i ((S1)+(S2)). We must also fill the leaves of exactly one edge-gadget in each edge-block ((S1)+(S3)). The transformation works by sliding tokens through green connector nodes. To consume at most $2n \cdot \kappa(\kappa - 1)$ token slides (equivalently, two token slides per token), we must for each pair $\{i, j\}$, slide n tokens from the selected vertex-gadget in the vertex-block for i to the selected edge-gadget in edge-block for $\{i, j\}$ through adjacent green connector nodes. Similarly, we must slide n tokens from the selected vertex-gadget in the vertex-block for j to the same edge-gadget through adjacent green connector nodes. This completely fills the $2n$ leaves of that edge-gadget and empties the selected vertex- and edge-gadgets.

It is easy to see that the reduction runs in polynomial time with respect to the number of vertices in G . We now present the lemmas necessary to complete the proof and start by

showing that the modulator to stars number is a function of κ .

Lemma 129. $ms(H) \leq 5\binom{\kappa}{2} + \kappa$.

Proof of Lemma 129. Let M be the set containing:

- for each unordered pair of color classes V_i and V_j , the green connector nodes $\text{index}(i, \{i, j\})$, $\text{remainder}(i, \{i, j\})$, $\text{index}(j, \{i, j\})$, $\text{remainder}(j, \{i, j\})$, and the gray edge block node $b_{i,j}$;
- for each color class V_i , the teal vertex block node b_i .

Then $G - M$ is a disjoint union of stars, since removing M from G leaves only the vertex-gadgets and edge-gadgets, each of which forms a star. $\square$

We then prove correctness via the following lemmas.

Lemma 130. *If (G, κ) is a yes-instance of MULTICOLORED CLIQUE, then $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY.*

Proof of Lemma 130. Let $K \subseteq V(G)$ be a multicolored clique in G of size κ . Let us now provide a transformation from the initial set of tokens S to a valid final configuration in b token sliding steps. We repeat the following procedure for each unordered pair of color classes V_i and V_j .

For V_i and V_j , let $\iota(u) = x \in [n]$ (index of u is x) be such that u is the vertex in $K \cap V_i$ and $\iota(v) = y \in [n]$ (index of v is y) be such that v is the vertex in $K \cap V_j$. We know that the edge uv exists since K is a clique. We slide tokens as follows.

- From the vertex-gadget for u in the vertex-block for i :
 - we slide x tokens from the first x leaves in the group corresponding to color class V_j to $\text{index}(i, \{i, j\})$ and then to the first x leaves of the edge-gadget for uv in the edge-block for $\{i, j\}$,
 - we slide $n - x$ tokens from the remaining $n - x$ leaves in the same group to $\text{remainder}(i, \{i, j\})$ and then to the next $n - x$ leaves of the edge-gadget for uv .
- From the vertex-gadget for v in vertex-block for j :

- we slide y tokens from the first y leaves in the group corresponding to color class V_i to $\text{index}(j, \{i, j\})$ and then to the next y leaves of the edge-gadget for uv in the edge-block for $\{i, j\}$;
- we slide $n - y$ tokens from the remaining $n - y$ leaves in the same group to $\text{remainder}(j, \{i, j\})$ and then to the final $n - y$ leaves of the edge-gadget for uv .

Let T be the final configuration. Then it is easy to see that T satisfies conditions (S1), (S2), and (S3), since in each vertex-block, exactly one vertex-gadget loses all its tokens ($n(\kappa - 1)$ tokens in total), in each edge-block exactly one edge-gadget gains tokens on all its leaves ($2n$ tokens), all tokens are on star leaves, and the total number of token slides used during the transformation is $2n(\kappa - 1)\kappa$ (for each $i < j \in [\kappa]$, $2n$ tokens slide exactly twice). Hence, $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY. $\square$

To prove the backward direction, we will need the following lemma.

Lemma 131. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, then*

- i. each token that moves is moved exactly two token sliding steps and passes through a green connector node,*
- ii. for each edge-block, the final configuration T contains $2n$ leaves of exactly one edge-gadget in the edge-block, and*
- iii. exactly $n(\kappa - 1)$ tokens slide from exactly one vertex-gadget in each vertex-block.*

Proof of Lemma 131. Since T is a valid final configuration and satisfies Condition S3, each edge-block must contain $2n$ tokens on the leaves of exactly one edge-gadget, which proves (ii). Since the leaves of each edge-gadget are not adjacent to any node in S , each token that ends up on a leaf of this edge-gadget requires at least two token slides. So, at least $4n$ token slides are needed for each edge-block. Since $b = 2n\kappa(\kappa - 1) = 4n\binom{\kappa}{2}$ and the above argument holds for all $\binom{\kappa}{2}$ edge-blocks, if there is a solution, every token that moves slides exactly twice. Since green connector nodes are the only nodes adjacent to both leaves of vertex-gadgets and leaves of edge-gadgets, the intermediate positions of all moved tokens must be green connector nodes, which proves (i). From Condition S2, for each vertex-block, exactly one vertex-gadget releases all its tokens (which are $n(\kappa - 1)$ tokens), which proves (iii). $\square$

Lemma 132. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, then (G, κ) is a yes-instance of MULTICOLORED CLIQUE.*

Proof of Lemma 132. Let $(T, \vec{T})$ be a solution to $((H, \mathcal{C}), S, b, \phi)$. From (S1) no block, root, or connector node contains a token. From Lemma 131, for each edge-block, exactly $2n$ leaves of one edge-gadget must contain tokens in the final configuration T . Each of these tokens slides through one of the green connector nodes and uses exactly two token slides. So each of the $2n$ tokens that end up in the edge-block for $\{i, j\}$ must come from one of the green connector nodes $\text{index}(i, \{i, j\})$, $\text{remainder}(i, \{i, j\})$, $\text{index}(j, \{i, j\})$, or $\text{remainder}(j, \{i, j\})$ after leaving one of the vertex-blocks for i or j (since these green connectors are not adjacent to leaves from other vertex-blocks).

By Lemma 131, there exist vertices $u \in V_i$ and $v \in V_j$ such that the vertex-gadget for u and the vertex-gadget for v are completely emptied of tokens (and all other vertex-gadgets in the vertex-blocks for i or j retain all their tokens). Similarly, there exists an edge $u'v'$ with $u' \in V_i$ and $v' \in V_j$ such that all leaves of the edge-gadget for e are in T (and no other edge-gadget in the edge-block for $\{i, j\}$ gets a token).

We show that $u = u'$ and $v = v'$. Let $\iota(u) = x$ and $\iota(v) = y$. Recall from the construction that the vertex-gadget for u has exactly n leaves in the group corresponding to color class V_j , where the first x leaves are adjacent only to $\text{index}(i, \{i, j\})$ among all green connector nodes, and the remaining $n - x$ leaves are adjacent only to $\text{remainder}(i, \{i, j\})$ among all green connector nodes. Similarly, the vertex-gadget for v has exactly n leaves in the group corresponding to color class V_i , where the first y leaves are adjacent only to $\text{index}(j, \{i, j\})$ among all green connector nodes, and the remaining $n - y$ leaves are adjacent only to $\text{remainder}(j, \{i, j\})$ among all green connector nodes.

Since $2n$ tokens must reach the edge-gadget for $u'v'$ through the green connector nodes from the vertex-gadgets for u and v using exactly two token slides each, each token from these vertex-gadgets moves to its unique adjacent green connector node during the first token slide. Thus, during the transformation, x tokens from the vertex-gadget for u slide to $\text{index}(i, \{i, j\})$ and $n - x$ tokens slide to $\text{remainder}(i, \{i, j\})$. Similarly, y tokens from the vertex-gadget for v slide to $\text{index}(j, \{i, j\})$ and $n - y$ tokens slide to $\text{remainder}(j, \{i, j\})$.

All these tokens must next slide to the same edge-gadget for $u'v'$. Let $\iota(u') = x'$ and $\iota(v') = y'$. The edge-gadget for $u'v'$ has x' leaves adjacent to $\text{index}(i, \{i, j\})$, $n - x'$ leaves adjacent to $\text{remainder}(i, \{i, j\})$, y' leaves adjacent to $\text{index}(j, \{i, j\})$, and $n - y'$ leaves adjacent to $\text{remainder}(j, \{i, j\})$. For all $2n$ tokens to successfully slide from the green connector nodes to fill all leaves of this edge-gadget using only one additional token slide per token, we must have $x = x'$ and $y = y'$. Thus $u = u'$ and $v = v'$, which means the edge uv exists in G .

This equality holds for every unordered pair of color classes. Thus, for each $i < j \in [\kappa]$, the vertices whose vertex-gadgets are emptied in the vertex-block for i and the vertex-block

for j are adjacent in G , which establishes a multicolored clique in (G, κ) . $\square$

7.2.2 Hardness of FO-Discovery for Modulator to Paths Number

We now present the hardness for the parameter modulator to paths number. More formally, the goal of this section is to prove the following:

Theorem 133. *FO-DISCOVERY is $W[1]$ -hard when parameterized by modulator to paths number.*

The rest of this subsection is devoted to the proof of Theorem 133. Let (G, κ) be an instance of the MULTICOLORED CLIQUE problem. We consider the graph $(H, \mathcal{C})$ described at the beginning of Section 7.2 where the vertex-blocks and edge-blocks of H are path vertex-blocks and path edge-blocks, the set S consisting of all nodes of all path vertex-gadgets, and the budget $b = 2n(\kappa - 1)\kappa$. We complete $((H, \mathcal{C}), S, b)$ with an FO-formula ϕ to create an instance of FO-DISCOVERY. We form the FO-formula $\phi(X)$ to express that the set X satisfies the following conditions (different from those in the modulator to stars reduction):

P1. No colored node contains a token. (That is, the block nodes and the connector nodes do not contain tokens.)

P2. For each teal (vertex block) node, there exists at least one adjacent uncolored node (in a path vertex-gadget) that is token-free.

P3. For each gray (edge block) node, there exists at least one adjacent uncolored node (in a path edge-gadget) that contains a token.

P4. Every uncolored node v (in a gadget) contains a token if and only if all uncolored nodes (in a gadget) at distance at most two from v in $H[V \setminus (C_2 \cup C_4 \cup C_5)]$ also contain tokens. (That is, a node of a gadget contains a token if and only if the gadget nodes at distance two from it in the subgraph excluding all block and connector nodes contain tokens.)

The construction of these expressions in FO is presented in Appendix A.1.2.

Before proving the theorem, let us explain some of the intuition behind the conditions. Condition P4 ensures that in each path gadget, either all nodes contain tokens or all nodes are token-free. Combined with conditions (P2) and (P3), this means that in each vertex-block, exactly one path vertex-gadget becomes token-free, while in each edge-block, exactly one path edge-gadget becomes filled with tokens.

It is easy to see that the reduction runs in polynomial time with respect to the number of vertices in G . We now show that the modulator to paths number is bounded.

Lemma 134. $mp(H) \leq 5\binom{\kappa}{2} + \kappa$.

Proof of Lemma 134. Let M be the set containing:

- for each unordered pair of color classes V_i and V_j , the green connector nodes $\text{index}(i, \{i, j\})$, $\text{remainder}(i, \{i, j\})$, $\text{index}(j, \{i, j\})$, $\text{remainder}(j, \{i, j\})$, and the gray edge block node $b_{i,j}$;
- for each color class V_i , the teal vertex block node b_i .

Then $G - M$ is a disjoint union of paths, since removing M from G leaves only the vertex-gadgets and edge-gadgets, each of which forms a path. $\square$

Lemma 135. *If (G, κ) is a yes-instance of MULTICOLORED CLIQUE, then $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY.*

Proof of Lemma 135. The proof follows the same transformation as in Lemma 130. The token movements use only edges between the gadgets nodes and the green connector nodes, which are identical in both constructions. The final configuration T satisfies conditions (P1)-(P4): no colored nodes contain tokens, each vertex-block has one empty path vertex-gadget, each edge-block has one full path edge-gadget, and Condition P4 is satisfied since tokens fill or empty entire paths. $\square$

To prove the backward direction, we need the following lemma.

Lemma 136. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, then*

- i. each token that moves is moved exactly two token sliding steps and passes through a green connector node,*
- ii. for each edge-block, the final configuration T contains tokens on all $2n$ nodes of exactly one edge-gadget in the edge-block, and*
- iii. exactly $n(\kappa - 1)$ tokens slide from exactly one vertex-gadget in each vertex-block.*

Proof of Lemma 136. By Condition P4, one node in a path contains a token if and only if all nodes in that path contain tokens. From conditions (P3) and (P4), for each edge-block, at least one path edge-gadget must be completely filled with tokens. Thus, exactly $2n$ tokens must be placed on one path edge-gadget.

Since the nodes of edge-gadgets are not adjacent to any nodes in S , each token that ends up on an edge-gadget requires at least two token slides. So, at least $4n$ token slides are needed for each edge-block. Since $b = 2n(\kappa - 1)\kappa = 4n\binom{\kappa}{2}$ and $4n$ token slides are needed for all $\binom{\kappa}{2}$ edge-blocks, every token that moves slides exactly twice and every edge-block gets to fill exactly one of its path edge-gadgets, which proves (ii). Since green connector nodes are the only nodes adjacent to both nodes of vertex-gadgets and nodes of edge-gadgets, all moved tokens must pass through green connector nodes, which proves (i).

From conditions (P2) and (P4), for each vertex-block, at least one path vertex-gadget must be completely token-free. Since each path vertex-gadget initially contains $n(\kappa - 1)$ tokens, exactly this many tokens must slide from one vertex-gadget, which proves (iii). $\square$

Lemma 137. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, then (G, κ) is a yes-instance of MULTICOLORED CLIQUE.*

Proof of Lemma 137. The proof follows the same argument as in Lemma 132, using Lemma 136 instead of Lemma 131. The key observation is that Condition P4 ensures that entire paths are either filled or emptied, maintaining the same token movement pattern as in the star case. $\square$

This completes the proof of Theorem 133.

7.3 Hardness of FO-Discovery for Twin Cover Number

This section presents the proof of the following theorem.

Theorem 138. *FO-DISCOVERY is $W[1]$ -hard when parameterized by twin cover number.*

Recall from Section 3.6 of the preliminaries that a *twin cover* of a graph G is a set $M \subseteq V(G)$ whose deletion leaves a disjoint union of cliques such that the vertices of each clique are twins. Two vertices $u, v \in V(G)$ are *twins* if $N(u) = N(v)$ or $N[u] = N[v]$. The *twin cover number* is the size of a minimum size twin cover.

To prove Theorem 138, we reduce from the PLANAR ARC SUPPLY problem. In this problem, we are given a (planar¹) digraph D with $|V(D)| + |A(D)| = \kappa$, no loops, and

¹The graph class is not relevant for the proof.

no double arcs. Each node $u \in V(D)$ has a demand $\delta(u)$ and each arc $uv \in A(D)$ has a list $L_{uv} = \{(x_{uv}^1, y_{uv}^1), (x_{uv}^2, y_{uv}^2), \dots, (x_{uv}^\ell, y_{uv}^\ell)\}$ of integer pairs, called the *supply pairs* of uv . We let $\mathcal{L}$ represent the supply lists of all arcs and $\lambda : A(D) \rightarrow \mathcal{L}$ be a function that maps each arc in D to a list in $\mathcal{L}$. Informally, we are asked to pick from the list of each arc $uv \in A(D)$ a supply pair (x_{uv}^i, y_{uv}^i) and send a supply of x_{uv}^i to u and a supply of y_{uv}^i to v such that for every vertex v , the sum of the supplies from the arcs incident to v satisfies exactly its demand. More formally, the task is to decide whether there is a function $\rho_x : A(D) \rightarrow \mathbb{N}$ and a function $\rho_y : A(D) \rightarrow \mathbb{N}$ such that for every arc $uv \in A(D)$ we have $(\rho_x(uv), \rho_y(uv)) \in L_{uv}$ and for every vertex $u \in V(D)$ we have:

$$\delta(u) = \sum_{v \in N^+(u)} \rho_x(uv) + \sum_{v \in N^-(u)} \rho_y(vu).$$

The problem is W[1]-hard when parameterized by κ [36, Section 3.2] even when the integers are given in unary, and vertices have either degree two or degree four.

Let (D, δ, λ) be an instance of PLANAR ARC SUPPLY. At a high level, the instance $((H, \mathcal{C}), S, b, \phi)$ of FO-DISCOVERY will have a *demand clique* for each vertex (initially with no tokens), a *reservoir clique* for each arc (initially full of tokens), and two *supply cliques* for each tuple in each arc's supply list in D (see Figure 7.7a). The key insight of the reduction is that token movements in H will encode the selection of supply pairs from the lists in $\mathcal{L}$. For clarity, we also refer to *vertices* in PLANAR ARC SUPPLY and *nodes* in our construction.

The twin cover structure acts as a “control skeleton” that connects the different cliques: we create a twin cover node for each original vertex and two twin cover nodes for each arc in D , ensuring that the resulting graph has a bounded twin cover number. Each vertex $u \in V(D)$ gets a corresponding twin cover *vertex node* connected to its demand clique of size $\delta(u)$, while each arc $uv \in A(D)$ gets two twin cover *arc nodes*, one *u-arc node for uv* and one *v-arc node for uv*. The *u-arc node for uv* is connected to supply cliques, each encoding the first component of a distinct supply pair (the values of x) in L_{uv} . These supply cliques are also connected to the vertex node for u and we refer to them as the *u-supply cliques for uv*. The *v-arc node for uv* is connected to supply cliques, each encoding the second component of a distinct supply pair (the values of y) in L_{uv} . These supply cliques are also connected to the vertex node for v and we refer to them as the *v-supply cliques for uv*. Both arc nodes for uv are connected to the reservoir clique for uv .

Each supply clique contains two types of nodes: *supply nodes* that initially contain tokens (whose number equals the corresponding component value in the supply pair) and empty *index nodes* (whose number encodes the index of the tuple within the supply list).

The FO-formula ensures that for each arc, we must “activate” exactly one supply pair by expressing that for each arc node, exactly one of the adjacent supply cliques empties its supply nodes and fills its index nodes with tokens. The FO-formula additionally specifies that the reservoir cliques are emptied of tokens and the demand cliques are full of tokens.

The crux of our reduction lies in a numerical encoding scheme (see Figure 7.7b). Each reservoir clique for an arc $uv \in A(D)$ contains Σ^2 tokens (where Σ is larger than the sum of all demand and supply values), while the number of index nodes in the supply cliques for uv and u and the number of index nodes in the supply cliques for uv and v uses a base- Σ representation: for the i th supply pair of the arc uv , the corresponding supply clique for uv and u has $\Sigma^2 - \Sigma \cdot (i - 1)$ index nodes and the corresponding supply clique for uv and v has $\Sigma \cdot (i - 1)$ index nodes. This creates a useful mathematical property: if forced to split between only two supply cliques, one connected to the arc node for u and one connected to the arc node for v , the Σ^2 tokens from a reservoir clique can only fill the index nodes of a matching pair of supply cliques (corresponding to a tuple in L_{uv}) whose index node counts sum up to exactly Σ^2 .

This numerical constraint, combined with our tight budget of only two token slides per displaced token, creates a forced choreography of token movements. Tokens must flow into a demand clique for a vertex $u \in V(D)$ from at most one u -supply clique for each incident arc to u . These token slides will amount to two times the sum of all demands in the instance. Additionally, with this budget constraint, tokens in a reservoir clique must first fill the index nodes of exactly one pair of supply cliques that correspond to a tuple in L_{uv} . Once the index nodes are filled, the tokens on the supply nodes of these activated cliques can be displaced and must move to their corresponding demand cliques. Any deviation from this pattern either violates the formula or exceeds the budget. Thus, valid token movements exist if and only if we can select supply pairs whose values exactly satisfy all vertex demands and (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY.

In the following description of an instance $((H, C), S, b, \phi)$ of FO-DISCOVERY, we use specific colors both for greater clarity and for use in the FO-formula.

As described above, we set Σ to be twice the sum of all integers in the input (that is, the sum of all demands and all integers appearing inside the tuples for all tuples). We form the instance $((H, C), S, b, \phi)$ of FO-DISCOVERY as follows. See Figure 3.13b for a sketch of the construction.

Twin cover. For each $u \in V(D)$ we create a *vertex node* and for each arc $uv \in A(D)$ we create two *arc nodes*, one u -arc node for uv and one v -arc node for uv .

We assign the following colors to the nodes in the twin cover:

- purple (C_1) to all vertex nodes, and
- black (C_2) to all arc nodes.

Demand cliques. For each $u \in V(D)$, we form a *demand clique* for u of size $\delta(u)$, whose nodes are adjacent to the vertex node for u .

We assign yellow (C_3) to all nodes in the demand cliques.

Supply cliques. For each arc uv such that L_{uv} has ℓ pairs, we create, for each $i \in [\ell]$, a *u-supply clique* for uv with x_{uv}^i supply nodes and $\Sigma^2 - \Sigma \cdot (i - 1)$ index nodes. We also create, for each $i \in [\ell]$, a *v-supply clique* for uv with y_{uv}^i supply nodes and $\Sigma \cdot (i - 1)$ index nodes.

We assign the following colors to the nodes in the supply cliques:

- orange (C_4) to all supply nodes, and
- teal (C_5) to all index nodes.

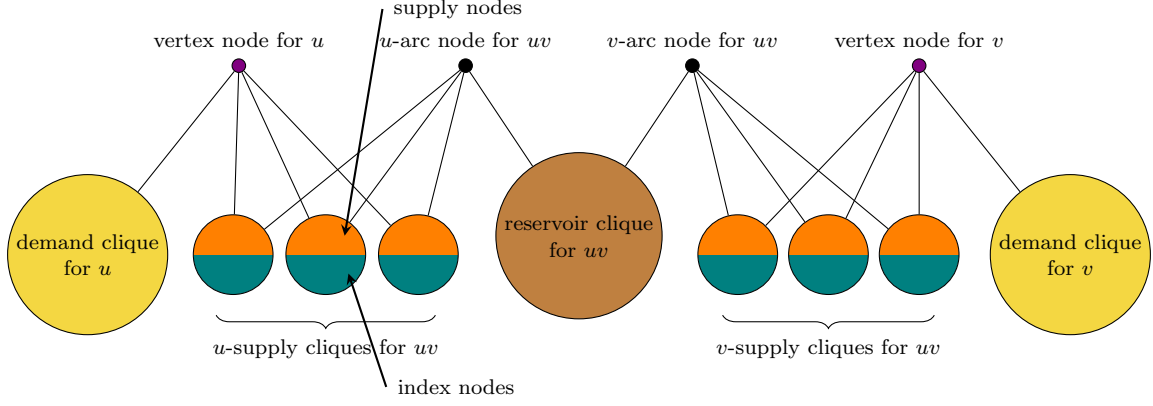
Reservoir cliques. For each arc $uv \in A(D)$, we form a *reservoir clique* for uv that contains Σ^2 nodes and make its nodes adjacent to the arc node for u and the arc node for v .

We assign brown (C_6) to all nodes in the reservoir cliques.

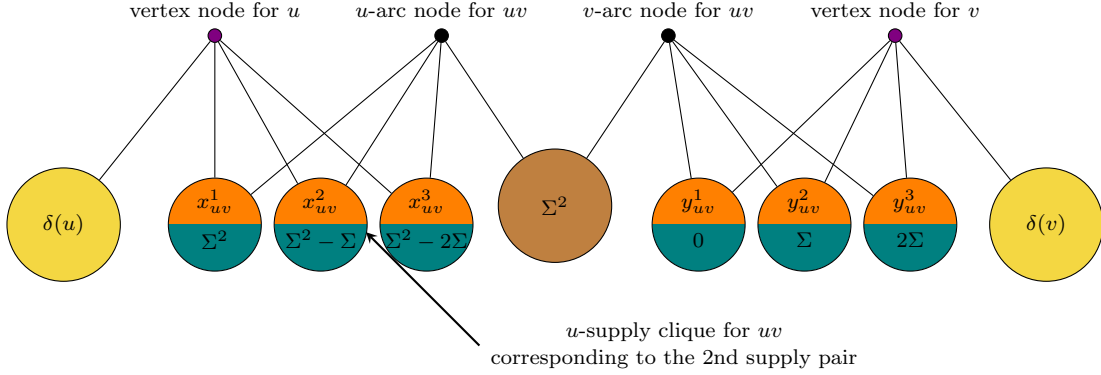
Orange supply nodes and brown nodes of reservoir cliques contain tokens. Let Δ be the sum of the demands of all the vertices in D (equivalently, the total size of all the demand cliques in H). We set the budget to $2\Delta + |A(D)| \cdot 2\Sigma^2$. The first component (2Δ) accounts for moving tokens from supply nodes to demand cliques (each token making two token slides: from its initial supply clique to a twin cover arc node, then to a demand clique). The second component ($|A(D)| \cdot 2\Sigma^2$) accounts for the Σ^2 tokens in each reservoir clique, where each token also makes two token slides to go from the reservoir clique to fill the index nodes in the selected supply cliques pair.

Our twin cover is made up of one vertex node for each vertex $u \in V(D)$ and two arc nodes for each arc $uv \in A(D)$. Deleting the nodes in the twin cover leaves a graph with only disjoint cliques. Additionally, each of the nodes in the twin cover is adjacent to all or none of the nodes in each clique. Thus, the twin cover number is at most $\kappa + 2\binom{\kappa}{2}$.

To complete the construction of the FO-DISCOVERY instance $((H, C), S, b, \phi)$, we form the FO-formula $\phi(X)$ to express that X satisfies the following conditions:



(a)



(b)

Figure 7.7: Part of the colored graph $(H, \mathcal{C})$ formed by the reduction corresponding to the twin cover number. It shows the part of the twin cover and the cliques that correspond to two vertices $u, v \in V(D)$ and the edge $uv \in A(D)$ with $|L_{uv}| = 3$. A line from a vertex or arc node to a clique means that the node is adjacent to all nodes of the clique. The nodes of the demand clique are yellow and those of the reservoir clique are brown. Supply nodes are orange, index nodes are teal, vertex nodes are purple, and arc nodes are black. (a) Since $|L_{uv}| = 3$, we have three u -supply cliques for uv and three v -supply cliques for uv . (b) Each u -supply clique for uv corresponding to the i th supply pair contains $\Sigma^2 - \Sigma \cdot (i - 1)$ teal (C_5) index nodes and x_{uv}^i orange (C_4) supply nodes. Each v -supply clique for uv corresponding to the i th supply pair contains $\Sigma \cdot (i - 1)$ teal index nodes and y_{uv}^i orange supply nodes.

T1. Each yellow (demand clique) node contains a token.

T2. The brown (reservoir clique) nodes do not contain tokens.

T3. For each pair of purple (vertex) and black (arc) nodes, exactly one adjacent (supply) clique containing at least one orange (supply) node must be such that all its teal (index) nodes contain tokens and all its orange (supply) nodes do not. For all other adjacent (supply) cliques of this type, their orange (supply) nodes contain tokens, and their teal (index) nodes do not. (That is, for every arc $uv \in A(D)$ there exists a unique u -supply clique for uv and a unique v -supply clique for uv that are modified: all tokens are removed from the supply nodes, and every index node is filled with a token.)

T4. The purple (vertex) and black (arc) nodes do not contain tokens.

The construction of these expressions in FO is given in Appendix A.2 to not disrupt the flow of the thesis. The fact that the twin cover number is at most $\kappa + 2\binom{\kappa}{2}$ and that the construction runs in polynomial time ensures that we obtain a valid FPT-reduction. We start by presenting the forward direction of the proof of Theorem 138.

Lemma 139. *If (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY, then $((H, C), S, b, \phi)$ is a yes-instance of FO-DISCOVERY.*

Proof of Lemma 139. Let ρ_x and ρ_y be a solution of (D, δ, λ) . For each arc $uv \in A(D)$, let (x_{uv}^i, y_{uv}^i) be such that $\rho_x(uv) = x_{uv}^i$ and $\rho_y(uv) = y_{uv}^i$.

Let us now describe a solution of $((H, C), S, b, \phi)$. We slide the x_{uv}^i tokens from the orange supply nodes of the u -supply clique for uv corresponding to the i th supply pair for uv toward the yellow nodes of the demand clique for u and the y_{uv}^i tokens from the orange supply nodes of the v -supply clique for uv corresponding to the i th supply pair for uv toward the yellow nodes of the demand clique for v . In total, for all arcs uv , this consumes 2Δ token slides. We then slide $\Sigma^2 - \Sigma \cdot (i - 1)$ tokens from the reservoir clique for uv to the teal index nodes of the u -supply clique for uv corresponding to the i th supply pair for uv and $\Sigma \cdot (i - 1)$ tokens to the teal index nodes of the v -supply clique for uv corresponding to the i th supply pair for uv . This consumes $|A(D)| \cdot 2\Sigma^2$ token slides.

Given that ρ_x and ρ_y are a solution, Condition T1 is satisfied. Since $\Sigma^2 - \Sigma \cdot (i - 1) + \Sigma \cdot (i - 1) = \Sigma^2$, Condition T2 is also satisfied. Condition T3 is satisfied since we chose the supply cliques for uv to correspond to one supply tuple in the supply list for uv and moved the tokens at the nodes of the reservoir clique for uv to fill the teal index nodes of the chosen supply cliques and moved the tokens at the orange supply nodes out of those chosen supply cliques. Finally, no purple vertex or black arc node is in T and Condition T4 is satisfied. Thus, we find that $((H, C), S, b, \phi)$ is a yes-instance, as needed. $\square$

The following lemma shows the backward direction of the proof of Theorem 138.

Lemma 140. *If $((H, C), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, then (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY.*

Proof of Lemma 140. Assume $((H, C), S, b, \phi)$ is a yes-instance of FO-DISCOVERY, and let $\mathcal{T}$ be a transformation from the initial set of tokens to a valid final configuration. Then, at least $|A(D)| \cdot 2\Sigma^2$ token slides are needed to empty the brown nodes of the reservoir cliques and satisfy conditions (T2) and (T4). Since the order of the moves of different tokens does not impact the validity of a solution, we can assume that these moves are performed first in the transformation and let us call S' the resulting configuration of tokens. Note that all nodes in S' are in supply cliques.

Since the total budget is $2\Delta + |A(D)| \cdot 2\Sigma^2$, there exists a transformation from S' to a valid final configuration in at most 2Δ token slides. Since no token of S' is incident to a demand clique, at least 2Δ token slides are needed in order to fill the Δ yellow nodes of demand cliques and satisfy Condition T1. Thus, from S' , exactly Δ tokens have to move and require exactly two token slides each to fill the demand cliques and these are the only token slides that remain.

The final configuration satisfies conditions (T3) and (T4). Let us consider the supply cliques to which the tokens initially in the reservoir cliques went. By Condition T3, the final configuration is modified compared to the initial configuration only on one u -supply clique for uv and one v -supply clique for uv for every arc $uv \in A(D)$. Thus, for every arc $uv \in A(D)$, there exist one chosen u -supply clique for uv (say, corresponding to the i th supply pair for uv) and one chosen v -supply clique for uv (say, corresponding to the j th supply pair for uv) such that more than Δ tokens of the reservoir clique for uv slide and stay on (as at most Δ of Σ^2 tokens can slide toward the demand cliques for u and v).

Moreover, since tokens of a valid final configuration lie on different nodes, there are at most Δ tokens that can move from the chosen u -supply clique for uv and the chosen v -supply clique for uv to demand cliques. Since all teal index nodes in the chosen u -supply clique for uv and the chosen v -supply clique for uv have to contain tokens in the final configuration and since all those tokens must come from the reservoir clique for uv , $\Sigma^2 - \Sigma \cdot (i - 1)$ of those tokens move to the chosen u -supply clique for uv and $\Sigma \cdot (j - 1)$ move to the chosen v -supply clique for uv . So we should have $\Sigma^2 \geq \Sigma^2 - \Sigma \cdot (i - 1) + \Sigma \cdot (j - 1)$ and thus $i \geq j$. Moreover, the total number of tokens that can move after S' is bounded by Δ , and the number of tokens that move from the reservoir clique for uv to the chosen supply cliques is $\Sigma^2 - (\Sigma^2 - \Sigma \cdot (i - 1) + \Sigma \cdot (j - 1)) = \Sigma \cdot (i - 1) - \Sigma \cdot (j - 1) \leq \Delta$. For Σ to be smaller than or equal to Δ , $i \leq j$. Thus, $i = j$ and all the tokens that move from

the reservoir clique for uv go to teal index nodes of the chosen u -supply clique for uv and the chosen v -supply clique for uv .

Condition T3 ensures that the tokens initially on the orange supply nodes of the chosen u -supply clique for uv and the chosen v -supply clique for uv have to move. Since every token that moves after S' can move exactly twice to go to the unique demand clique at distance two from it, all the tokens on the orange supply nodes of the chosen u -supply clique for uv have to go to the nodes of the demand clique for u and all the tokens on the orange supply nodes of the chosen v -supply clique for uv have to go to the nodes of the demand clique for v . This argument holds for every arc in the instance.

Since tokens lie on different nodes of the final configuration, for every $u \in A(D)$, the sum of the tokens going to the demand clique of u from the u -supply cliques for the arcs incident to u cannot be larger than $\delta(u)$. To conclude, let us prove that the sum has to be equal to $\delta(u)$. Assume by contradiction that some demand is not satisfied, i.e., (T1) is not satisfied for a vertex u after moving the tokens on the orange supply nodes. To transform the current set into a valid final configuration, additional tokens from a u -supply clique for some arc incident to u should go to the demand clique for u . By Condition T3, such tokens cannot be initially on an orange supply node of another u -supply clique. Additionally, as argued before such tokens cannot be tokens initially on a brown node in a reservoir clique as these tokens fill the teal index nodes and cannot slide further. So the demand of each vertex $u \in V(D)$ has to be exactly satisfied from the orange supply nodes of the chosen u -supply cliques for incident arcs, which completes the proof. $\square$

7.4 Hardness of MSO-Discovery for Bandwidth

The focus of this section is the proof of the following theorem.

Theorem 141. *MSO-DISCOVERY is $W[1]$ -hard when parameterized by bandwidth.*

Recall from Section 3.6 of the preliminaries that the *bandwidth* of a graph G is the minimum integer $bw(G)$ such that there exists a bijection $f : V(G) \rightarrow \{1, \dots, |V(G)|\}$ satisfying that for every edge $\{u, v\} \in E(G)$, $|f(u) - f(v)| \leq bw(G)$.

We use a reduction from the PLANAR ARC SUPPLY problem, starting with a slight modification to form the instance (D, δ, λ) . To ensure that all lists are of equal length, while maintaining the validity of the supply constraints, if a list is not of the maximum length t , we add as many copies of the first tuple as needed to increase the length to t . We then create for each vertex $u \in V(D)$ a vertex-gadget H_u and for each arc $uv \in A(D)$

an arc-gadget H_{uv} . For clarity, we also refer to *vertices* in the instance of PLANAR ARC SUPPLY and *nodes* in our construction.

At a high level, we can view each arc-gadget and each vertex-gadget as a “ladder”, with two long paths forming the *side rails* and shorter paths forming the *rungs* that connect the side rails. We will refer to the “top” and “bottom” of a side rail, consistent with how one might draw an image of a ladder. The two side rails of an arc-gadget H_{uv} correspond to the two endpoints u and v of the arc uv . In a vertex-gadget, the *isolated side rail* has no connections outside the vertex-gadget, whereas the *connected side rail* has connections to side rails of arc-gadgets associated with (at most four) incident arcs. When two nodes in a vertex-gadget are joined by a rung (that is, one on the isolated side rail and one on the connected side rail), we call them *rail neighbors*.

To represent the supply pairs, an arc-gadget H_{uv} contains one *sub-ladder* for every supply tuple (x, y) in L_{uv} , for a total of t sub-ladders. At the bottom of each sub-ladder is a rung. Above the rung, the side rail corresponding to u will contain x tokens, and the side rail corresponding to v will contain y tokens. All the tokens will be located on *token-bearing nodes*. The MSO-formula ensures that each arc-gadget empties exactly one sub-ladder. It also specifies that for each vertex u , a token be moved to each of the $\delta(u)$ *demand nodes* located at the top of the vertex-gadget (on the connected side rail), achievable in a yes-instance of PLANAR ARC SUPPLY by moving tokens from the arc-gadget sub-ladders associated with the chosen supply pairs. The tokens first move from the side rails of the arc-gadgets into the corresponding vertex-gadgets, and from there up to the demand nodes.

Also specified in the MSO-formula, the *rail neighbor condition* requires that for any node g on the connected side rail that is the neighbor of a token-bearing node h , there must be a token either on h or on the rail neighbor of g (located on the isolated side rail), which we call the *mirror* of h . Each vertex-gadget will contain a total of $t + 1$ sub-ladders, one for each supply pair, as well as one at the bottom of the ladder, containing $\delta(u)$ *reservoir nodes* that each hold a token. Thus, when the tokens on a side rail of a sub-ladder of an arc-gadget are emptied into the vertex-gadget and moved up to the demand nodes, the rail neighbor condition is satisfied by moving tokens to the mirrors of those token-bearing nodes from the reservoir nodes.

Crucially, no matter which sub-ladder we pick in an arc-gadget, we incur the costs of moving its tokens upward to the demand nodes in the vertex-gadgets and moving tokens from the reservoirs in the vertex-gadgets to the mirrors of now-empty token-bearing nodes. Since the two costs, per shifted token, sum up to the length of a vertex-gadget side rail, the total cost, per shifted token, will be the same regardless of which sub-ladder of the arc-gadget we choose. The budget is chosen so that tokens can only slide in the described

manner, implying that each vertex-gadget for a vertex $u \in V(D)$ receives exactly $\delta(u)$ tokens and does so by taking in the appropriate tokens (i.e. supplies) of the selected sub-ladders (i.e. selected supply pairs) of connected arc-gadgets.

In order to restrict the ways tokens can slide, we introduce *spacer nodes*, designed to increase the lengths of certain subpaths within gadgets. There are spacer nodes along the side rails of arc-gadgets in order for them to “match” the lengths of the side rails of vertex-gadgets, and there are also spacer nodes along rungs.

The length of side rails is chosen to ensure that the side rail of each sub-ladder has enough space to hold and receive tokens. We set Σ to be three times the sum of all integers in the input (that is, the sum of all demands and, for all tuples, all integers appearing inside the tuples), and to account for there being as many as four neighbors of a vertex, we set the length of a side rail of a sub-ladder to be 4Σ .

In the following description of an instance $((H, \mathcal{C}), S, b, \phi)$ of MSO-DISCOVERY, we use specific colors both for greater clarity and for use in the MSO-formula.

Vertex-gadget. Each vertex-gadget H_u consists of two paths of length $4\Sigma(t+1)$ (the *isolated side rail* and the *connected side rail*, or collectively, the *side rails*), connected by $4\Sigma(t+1)$ paths of length four (the *rungs*); the three intermediate nodes in each rung are all *spacer nodes*. We use the term *sub-ladder* to refer to each of the $t+1$ subgraphs formed by splitting each side rail into subpaths of length 4Σ . We number the sub-ladders, with sub-ladder 1 at the top and sub-ladder $t+1$ at the bottom.

The $\delta(u)$ nodes at the top of the connected side rail are *demand nodes* and the $\delta(u)$ nodes at the bottom of the isolated side rail are *reservoir nodes*. The bottom-most sub-ladder of the vertex-gadget, namely the sub-ladder $t+1$, contains the reservoir nodes and has no connections outside the vertex-gadget. The MSO-formula implies that all nodes of the sub-ladder $t+1$ of each vertex-gadget do not contain tokens. Thus, we show later that tokens must leave sub-ladder $t+1$ of a vertex-gadget for vertex u through the bottom-most isolated side rail node of sub-ladder t of H_u , which we therefore refer to as the *transit node* for u .

Since a vertex $u \in V(D)$ has degree at most four, a vertex-gadget will be connected to at most four arc-gadgets. To distinguish among the nodes involved in the four connections, we assign each a *type* (a , b , c , or d). We can view the 4Σ nodes in the side rail of a sub-ladder as being the interleaving of four paths of length Σ , one of each type. Starting at the bottom and moving to the top, each side rail consists of nodes of types a , b , c , d , a , b , c , d , and so on.

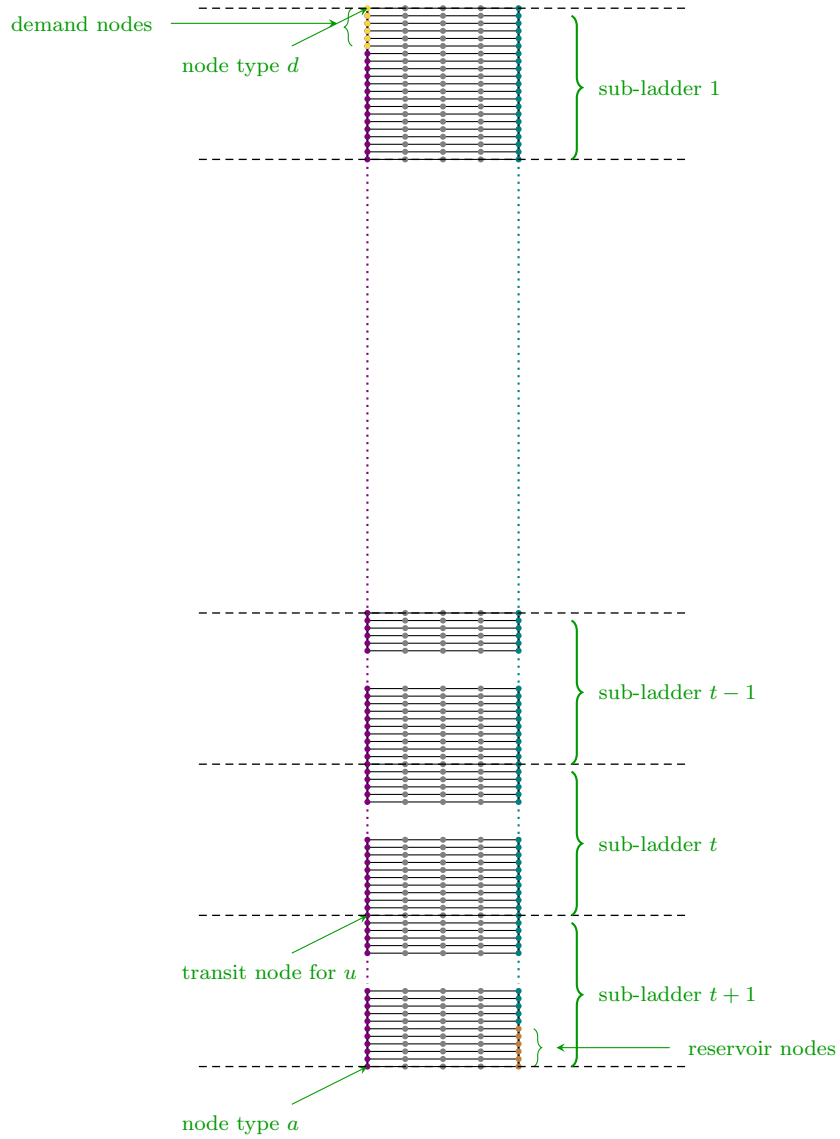


Figure 7.8: A sketch of the isolated and connected side rails for some vertex $u \in V(D)$ as discussed in Theorem 141. Demand nodes at the top of the connected side rail are colored yellow; reservoir nodes at the bottom of the isolated side rail are brown. Reservoir nodes contain tokens. The thin dotted lines indicate additional nodes that are omitted for clarity.

Initially, there is a token on each reservoir node, for a total of $\delta(u)$ tokens for H_u .

We assign the following colors to the nodes in the vertex-gadgets:

- yellow (C_1) to all demand nodes (including the topmost node in the connected side rail when $\delta(u) \neq 0$),

- purple (C_2) to all remaining connected side rail nodes,
- brown (C_3) to all reservoir nodes (including the bottom-most node in the connected side rail when $\delta(u) \neq 0$),
- teal (C_4) to all remaining isolated side rail nodes, and
- gray (C_5) to all spacer nodes.

Figure 7.8 illustrates a vertex-gadget, depicting sub-ladders, demand nodes, and reservoir nodes.

Arc-gadget. The arc-gadget H_{uv} for uv will contain one side rail for u and one for v . To represent the supply pairs, an arc-gadget H_{uv} contains one *sub-ladder* for each supply tuple (x, y) in L_{uv} (for a total of t sub-ladders since unlike in vertex-gadgets, here there are no reservoir nodes). The sub-ladders are formed by splitting each side rail into subpaths of length 4Σ . As in the vertex-gadgets, we number the sub-ladders, with sub-ladder 1 at the top and sub-ladder t at the bottom. Sub-ladder i corresponds to the supply pair i in the instance of PLANAR ARC SUPPLY.

In each sub-ladder, the bottom-most node in each side rail is a *rung node*. The *rung* connecting two rung nodes is a path of length six, with five intermediate spacer nodes.

To match the lengths of the paths between nodes of each type in the vertex-gadget, in a side rail of a sub-ladder, there are three spacer nodes between the rung node and the first *token-bearing node*, and three spacer nodes between each consecutive pair of *token-bearing nodes*. The total number of token-bearing nodes in a side rail of a sub-ladder will equal the number in the corresponding supply pair for that endpoint. All remaining nodes in the side rails of the sub-ladder will be spacer nodes.

Tokens are initially placed on all token-bearing nodes.

We assign the following colors for the nodes in the arc-gadgets:

- black (C_6) to all rung nodes,
- orange (C_7) to all token-bearing nodes, and
- gray (C_5) to all spacer nodes.

Figure 7.9 shows several sub-ladders in an arc-gadget.

We complete our construction of $(H, \mathcal{C})$ by connecting vertex-gadgets and arc-gadgets. For each vertex $u \in V(D)$, we assign a distinct type (a , b , c , or d) to each neighbor of u . Assuming, without loss of generality, that neighbor v of u is assigned type a , then there are edges only between (some) nodes of type a in H_u and nodes in H_{uv} . For each pair of sub-ladders with the same number, one in H_u and one in H_{uv} , we add an edge between

the rung node on the side rail for u in H_{uv} and the bottom-most node of type a in the sub-ladder on the connected side rail in H_u . Then, working up from the bottom upward in both sub-ladders, we connect the token-bearing nodes in H_{uv} one by one to distinct nodes of type a in the sub-ladder on the connected side rail in H_u .

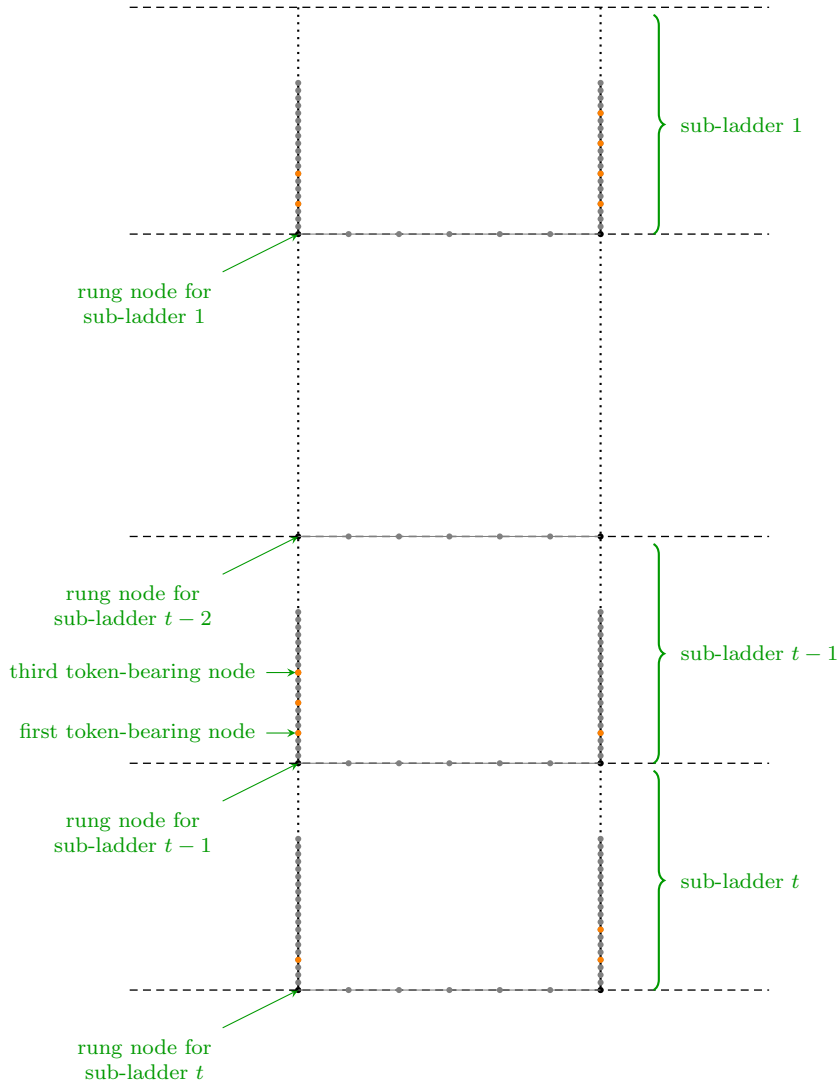


Figure 7.9: A sketch of the arc-gadget for some edge $uv \in A(D)$. Only the orange token-bearing nodes contain tokens. Here, the x and y values for the first supply tuple for uv are 2 and 4, respectively. The values for the supply tuple $t-1$ for uv are 3 and 1, respectively, and those for the supply tuple t for uv are 1 and 2, respectively.

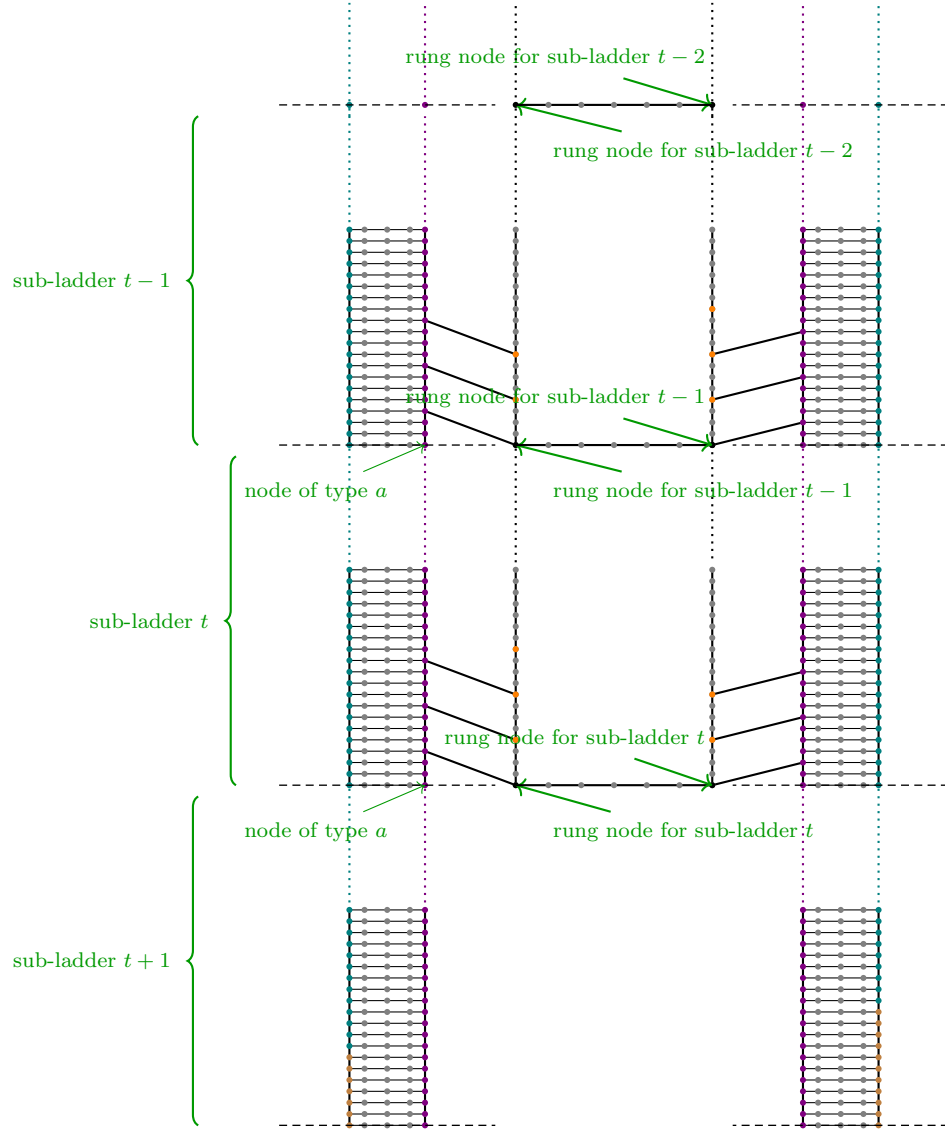


Figure 7.10: A sketch of how the connected side rail for some vertex $u \in V(D)$ and the connected side rail for some vertex $v \in V(D)$ connect to an arc-gadget for arc $uv \in A(D)$. Here v is assigned type d in the neighborhood of u , and u is assigned type c in the neighborhood of v . The thin dotted lines indicate additional nodes that are omitted for clarity.

Figure 7.10 presents a sketch of the attachments between edge-gadget H_{uv} and vertex-

gadgets H_u and H_v . Here v is assigned type d in the neighborhood of u , and u is assigned type c in the neighborhood of v .

We let Δ be the sum of all demands and then set $b = \Delta + \sum_{u \in V(D)} \delta(u) \cdot (4\Sigma(t+1) - \delta(u))$, sufficient for sliding Δ tokens from orange token-bearing nodes onto and then up the corresponding vertex-gadgets and then sliding tokens from the brown reservoir nodes to the teal mirrors of now empty orange token-bearing nodes. Each token that slides from an orange token-bearing node h onto a purple connected side rail node g in a vertex-gadget H_u , requires $4\Sigma(t+1) - \delta(u)$ token slides to slide the token towards a yellow demand node and slide a token from the brown reservoir nodes to the teal mirror of h .

We complete $((H, \mathcal{C}), S, b)$ with an MSO-formula $\phi(X)$ to create an instance of MSO-DISCOVERY. The formula $\phi(X)$ indicates that the set X satisfies the following conditions:

B1. The gray (spacer), the purple (non-demand connected side rail), and the brown (reservoir) nodes do not contain tokens.

B2. Each yellow (demand) node contains a token.

B3. A teal (isolated side rail) node contains a token if and only if an orange (token-bearing) node is at distance five from it and does not contain a token. (That is, the rail neighbor condition is satisfied.)

B4. If an orange (token-bearing) node contains a token, then the orange (token-bearing) nodes at distance four from it also do. Otherwise, they do not. (That is, for each sub-ladder, token-bearing nodes on one side rail either all contain tokens or none do.)

B5. For each connected component formed of (spacer, rung, and token-bearing) nodes of color gray, black, and orange, there exist exactly two orange (token-bearing) nodes not containing tokens, each at distance four from a black (rung) node, such that the black (rung) nodes are at a distance six from each other. No other black (rung) nodes at distance four from an orange (token-bearing) node not containing a token exist in the component. (That is, for each arc-gadget, we empty the orange token-bearing nodes each at distance four from the black rung nodes of exactly one sub-ladder. Combined with (B4), this condition implies that for each arc-gadget, we empty exactly one sub-ladder.)

The construction of these expressions in MSO is given in Appendix A.3. The formula cannot be expressed in FO since Condition B5 requires the quantification over a set that represents the connected component.

It is easy to see that the construction of the reduced instance can be carried out in polynomial time with respect to the number of vertices of D . We now show that the bandwidth of H is bounded above by a computable function of κ , the number of vertices

and edges in D , and then present the forward and backward directions of the proof of Theorem 141.

Lemma 142. $bw(H) \leq 66\kappa$.

Proof of Lemma 142. We describe a linear ordering of the vertices of H that achieves a bandwidth that is at most 66κ . First, we use the term *segment* to refer to the $\Sigma(t+1)$ subgraphs formed by splitting each side rail for each vertex-gadget and each arc-gadget into subpaths of length four. We number the subpaths, with segment 1 at the top and segment $\Sigma(t+1)$ (Σt for an arc-gadget) at the bottom. In an arc-gadget, each segment containing rung nodes also contains the rung connecting them. Other segments of the arc-gadget do not contain rungs.

For index $i \in [\Sigma t]$, we add to the ordering the nodes of the segment i for every vertex-gadget H_u for every vertex $u \in V(D)$ and every arc-gadget H_{uv} for every arc $uv \in A(D)$. Then, for each $i \in [\Sigma(t+1)] - [\Sigma t]$, we add to the ordering the nodes of the segment i for every vertex-gadget H_u .

It is easy to verify that every node in $V(H)$ is inserted exactly once. Moreover, each index $i \in [\Sigma(t+1)]$ adds at most 33κ nodes to the ordering (for each vertex-gadget, it adds four rungs made up of three gray nodes each, as well as four purple connected and four teal isolated side rail nodes, and for each arc-gadget, it adds at most eight side rail nodes and five nodes of a rung). Also, every node inserted at index i is adjacent only to the nodes inserted at indices $i-1$ or $i+1$, thus every node in the ordering is only adjacent to nodes at distance at most 66κ away from it in the ordering. This means by the definition of bandwidth that we have $bw(H) \leq 66\kappa$. $\square$

Lemma 143. *If (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY, then $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of MSO-DISCOVERY.*

Proof of Lemma 143. Let ρ_x and ρ_y be a solution for the instance. For an arc $uv \in A(D)$, let i be the index of the supply tuple in L_{uv} that sends its supply to the vertices u and v in D . We perform the following token slides:

1. We slide the tokens at the orange token-bearing nodes of sub-ladder i onto their purple neighboring nodes in vertex-gadgets H_u and H_v . Applying these token slides for every arc consumes Δ token slides.
2. For each vertex-gadget and for each purple connected side rail node g belonging to the gadget and now having a token, we slide the closest available token from the

brown reservoir nodes of the gadget towards the teal rail neighbor of g and slide the token on g toward the farthest empty yellow demand node. Since in Step 1 a vertex-gadget for vertex $u \in V(D)$ receives $\delta(u)$ tokens, we perform this step $\delta(u)$ times and require $4\Sigma(t+1) - \delta(u)$ token slides each time. Thus, applying these token slides for every vertex $u \in A(D)$ requires $\sum_{u \in V(D)} \delta(u) \cdot (4\Sigma(t+1) - \delta(u))$ token slides.

After performing the described token slides, for each vertex $u \in V(D)$, the yellow demand nodes contain tokens and the brown reservoir nodes do not, as each vertex-gadget, say for vertex u , receives exactly $\delta(u)$ tokens from the incident arc-gadgets onto its purple connected side rail nodes, satisfying Condition B2. Condition B1 is satisfied as no token is moved throughout the process to a gray spacer, purple connected side rail, or brown reservoir node. Condition B3 is satisfied by the token slides from Step 2 and since a token was placed on a teal isolated side rail node only when the teal isolated side rail node has an orange token-bearing node at distance five from it and this orange token-bearing node does not contain a token. Conditions (B4) and (B5) are satisfied since for every arc-gadget, we slide the tokens on all orange token-bearing nodes of exactly one sub-ladder of the gadget toward the vertex-gadgets. In total, the token slides we made require exactly the budget $b = \Delta + \sum_{u \in V(D)} \delta(u) \cdot (4\Sigma(t+1) - \delta(u))$. Thus, we find that $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of MSO-DISCOVERY. $\square$

To prove the backward direction, we first use the construction and the constraints on the formula to show that any solution can be transformed into a solution of a restricted form. We then make use of the restrictions on the solution to demonstrate that a solution to a yes-instance $((H, \mathcal{C}), S, b, \phi)$ of MSO-DISCOVERY can be used to form a solution to (D, δ, λ) .

We start by defining a *minimal solution* as one that has the minimum number of token slides and of all solutions that have the minimum number of token slides, one with the minimum number of tokens that slide. Since any solution must satisfy conditions (B1) through (B5), tokens are moved from selected orange token-bearing nodes and all brown reservoir nodes to selected teal isolated side rail nodes and all yellow demand nodes. When we eliminate any unneeded token slides, we obtain the following observation.

Observation 144. *In a minimal solution, each token that slides does so from an orange or brown node to a teal or yellow node.*

To better describe the movement of tokens, we define the *supernode* for a vertex $u \in V(D)$ to consist of the union of the following four sets of nodes:

- *the orange nodes of the supernode*, that is, each orange token-bearing node that is token-free in T and adjacent to a node of H_u ;
- *the teal nodes of the supernode*, that is, each teal isolated side rail node that is the mirror of an orange node of the supernode;
- *the yellow nodes of the supernode*, that is, all yellow demand nodes of H_u ; and
- *the brown nodes of the supernode*, that is, all brown reservoir nodes of H_u .

We can view the movement of a token from one node to another as either an *external arc* between supernodes (if the nodes are in different supernodes) or an *internal arc* within a supernode (if the nodes are in the same supernode). Our proof consists of showing first that there cannot be any external arcs (Lemma 150), and then that the movements within supernodes are restricted by the budget to follow certain types of paths (Lemma 151). The restricted types of paths can then be used form a solution to (D, δ, λ) .

We observe that the number of tokens in each supernode must remain unchanged, as the conditions ensure that the numbers of tokens removed from the brown and orange nodes of the supernode are equal to the numbers of tokens added to the yellow and teal nodes of the same supernode. Thus, if a supernode “loses” a token through an outgoing external arc, it must also “gain” a token through an incoming external arc, which leads to the following observation.

Observation 145. *In any supernode, the number of arcs leaving orange nodes is equal to the number of arcs arriving at teal nodes, and the number of arcs leaving brown nodes is equal to the number of arcs arriving at yellow nodes.*

We make use of Observation 145 to form sequences of arcs. An orange-teal arc where the orange and teal nodes are of the same supernode but not mirrors is called a *bad internal arc*. The arc *succeeding* a brown-teal or an orange-teal arc (where the orange and teal nodes are not mirrors) is the arc leaving from the teal node’s mirror. An arc *succeeding* a brown-yellow or orange-yellow arc is any external arc leaving from the supernode containing the yellow node.

Before proving Lemma 150, we further transform a minimal solution to obtain a few useful properties. In transforming a solution, we make use of simple modifications that change neither the validity of a solution nor the total number of required token slides. We will call a path traversed by a token between an orange node and a node of another color in the same supernode *friendly* if it first passes through the orange node’s purple connected

side rail neighbor. Similarly, we call a path traversed by a token to a teal node *friendly* if it last passes through the teal node's purple connected side rail neighbor and connecting rung.

Observation 146. *None of the following modifications changes the validity of a solution:*

1. *reordering token slides;*
2. *making a path traversed by a token between an orange node and a node of another color in the same supernode friendly;*
3. *making a path traversed by a token to a teal node after passing through an arc-gadget friendly; or*
4. *swapping the destinations of two tokens that pass through the same node.*

We define a *canonical solution* to be a minimal solution $(T, \vec{T})$ for the instance $((H, \mathcal{C}), S, b, \phi)$ of MSO-DISCOVERY such that:

1. for each vertex $u \in V(D)$, exactly $\delta(u)$ tokens slide from the brown reservoir nodes of the vertex-gadget for u to the transit node for u , with these token slides appearing first in $\vec{T}$,
2. if there is an external arc to a teal node and an orange-teal arc leaving from its mirror, that arc is an external arc,
3. there is no external arc to a teal node and an internal orange-yellow arc leaving from its mirror, and
4. if we have an internal orange-teal arc, the teal mirror (call it tm) of this arc's orange node participates in an internal brown-teal arc.

Lemma 147. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of MSO-DISCOVERY, then it has a canonical solution $(T, \vec{T})$.*

Proof of Lemma 147. Given conditions (B1) and (B3), we know that the tokens on the brown reservoir nodes of the vertex-gadget for u for a vertex $u \in V(D)$ do not end up on any node of the sub-ladder $t + 1$ for the vertex-gadget for u . Thus, these tokens will slide (or can be rerouted to slide without increasing the number of token slides) at least towards the transit node for u . It also follows from Observation 146, Point 1, that the

token slides that appear first in $\vec{T}$ are the token slides of the $\delta(u)$ tokens from the brown reservoir nodes of the vertex-gadget for u to the transit node for u , for each $u \in V(D)$. We have now satisfied Condition 1 of the canonical solution.

To see that we can satisfy Condition 2, suppose that we have an external arc to a teal node and a bad orange-teal internal arc leaving from its mirror. By Observation 146, points (2) and (3), we can make the paths traversed by the corresponding tokens friendly. Since both friendly paths traverse the same purple connected side rail node (that is the rail neighbor of the teal node), by Observation 146, Point 4, we can swap the destinations of the two tokens. We keep repeating the argument until we have a resulting external arc to a teal node with no bad orange-teal internal arc leaving from its mirror or we get a resulting external arc to a yellow node. Thus, Condition 2 now holds.

Now, suppose there is an external arc to a teal node and an internal orange-yellow arc leaving from its mirror. By Observation 146, points (2) and (3), we can make the paths traversed by the corresponding tokens friendly. Since both friendly paths traverse the same purple connected side rail node (that is the rail neighbor of the teal node), by Observation 146, Point 4, we can swap the destinations of the two tokens. The result is that Condition 3 is satisfied.

Finally, suppose we have an internal orange-teal arc and the teal mirror (call it tm) of this arc's orange node participates in an external arc, then this violates Condition 2. If tm participates in an internal orange-teal arc, by Observation 146, points (2) and (3), we can make the paths traversed by the tokens of both internal orange-teal arcs friendly. Since both friendly paths traverse the same purple connected side rail node (that is the rail neighbor of tm), by Observation 146, Point 4, we can swap the destinations of the two tokens. We keep repeating the argument until we have a that the teal node of the resulting internal orange-teal arc participates in an internal brown-teal arc, and thus satisfy Condition 4. $\square$

We show in Lemma 150 that any cycle of external arcs can be removed, resulting in a smaller number of token slides, and hence contradicting the minimality of a canonical solution. We do so by starting with an external arc and using it to build a cycle. The following observation is a direct consequence of Observation 145.

Observation 148. *If there is a single external arc in a canonical solution, there must be a cycle with one or more external arcs.*

We capture the movement of tokens between supernodes using an auxiliary graph $D^\pm$ containing supernodes and the external arcs between them. We show that no external

arcs are possible by demonstrating that any cycle in $D^\pm$ can be transformed into a smaller solution, violating the minimality of a canonical solution.

We construct cycles in $D^\pm$ by considering the colors of the nodes (of supernodes) from and to which the token is moving; by Observation 144 each arc is orange-teal, orange-yellow, brown-teal, or brown-yellow. Using a proof by contradiction, we show that if a canonical solution contains an external arc, we can find in $D^\pm$ a cycle consisting entirely of orange-teal external arcs or a cycle built out of smaller pieces. Specifically, we call a sequence of external arcs a *brown-yellow segment* if the first arc in the sequence starts at (i) a brown node or (ii) an orange node whose teal mirror participates in an internal brown-teal arc, or (iii) an orange node whose teal mirror participates in an internal orange-teal arc whose teal node participates in an internal brown-teal arc, and the last arc in the sequence ends at a yellow node; in the sequence, each arc is followed by a succeeding arc.

Observation 149. *If we can find a single brown-yellow segment in $D^\pm$, by Observation 148 we must find a cycle formed of one or more brown-yellow segments.*

For both types of cycles, we demonstrate that we can modify the token slides to produce a solution with a smaller number of slides, contradicting the assumption that the solution is canonical.

We are now ready to prove the following lemma:

Lemma 150. *No canonical solution contains an external arc.*

Proof of Lemma 150. Suppose to the contrary that a canonical solution contains an external arc. We consider two possible cases:

Case 1: There is a cycle in $D^\pm$ formed entirely of orange-teal arcs.

Since the arc succeeding an external orange-teal arc starts at the teal node's orange mirror and is external, the cycle contains matched pairs of mirrors. We create a new solution in which, for each pair, the token slides from the orange node to its mirror. Since we no longer require tokens to move along rungs of an arc-gadget (as was required in the solution that forms the cycle in $D^\pm$), we have reduced the number of token slides while preserving the number of tokens that slide. Doing so violates the assumption that the solution was canonical, as it was not minimal.

Case 2: There is no cycle in $D^\pm$ formed entirely of orange-teal arcs.

It suffices to demonstrate that we can use any of the other types of external arcs (brown-yellow, brown-teal, or orange-yellow) to form a brown-yellow segment. By Observation 149

we can then conclude that we have a cycle formed of brown-yellow segments (the cycle gives us matched pairs where each pair consists of a brown and a yellow node of a supernode in the cycle). Since in the cycle, each external arc to a teal node is followed by an external arc leaving from its mirror, the cycle also only contains matched pairs of mirrors. We can then form a solution using a smaller number of token slides by, for each brown-yellow pair, sending the token on the brown node to the matched yellow node in the same supernode, and for each pair of matched mirrors, send the token on the orange node to its matched teal mirror. Since we have decreased the number of token slides (by avoiding the traversal of rungs of at least one arc-gadget), the solution was not canonical.

To complete the proof, we show that in each of the following cases we can obtain a brown-yellow segment.

Case 2a: There is a brown-yellow arc in $D^\pm$.

In this case, the single arc is a brown-yellow segment.

Case 2b: There is a brown-teal arc in $D^\pm$.

If there is a brown-teal arc in $D^\pm$, then the succeeding arc starts at the orange mirror of the teal node. If the succeeding arc is an orange-yellow arc, we either have a brown-yellow segment (if the brown and yellow nodes are in different supernodes) or have completed a cycle (if the brown and yellow nodes are in the same supernode). Otherwise, we add the orange-teal arc, and continue adding succeeding orange-teal arcs until we eventually reach an arc with a yellow node (and hence form either a brown-yellow segment or a cycle).

Case 2c: There is an orange-yellow arc in $D^\pm$.

If there is an orange-yellow arc in $D^\pm$, it is the succeeding arc of either a brown-teal arc or an orange-teal arc. These arcs can be internal as well. As the case where we have a brown-teal external arc was covered in Case 2b, we consider the others. Consider an orange-teal external arc, and observe that it too is the succeeding arc of either a brown-teal arc or an orange-teal arc, which can be internal. We continue with the process of moving backward through external arcs until we find an internal orange-teal arc or an internal brown-teal arc. If we have an internal brown-teal arc, we form either a brown-yellow segment or a cycle. If we have an internal orange-teal arc, by definition of the canonical solution, the teal mirror of its orange node participates in an internal brown-teal arc. Thus, we also form either a brown-yellow segment or a cycle. $\square$

We are now ready to complete our proof of the backward direction.

Lemma 151. *If $((H, \mathcal{C}), S, b, \phi)$ is a yes-instance of MSO-DISCOVERY, then (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY.*

Proof of Lemma 151. Let $(T, \vec{T})$ be a canonical solution. From Lemma 150, we know that each token on an orange or brown node of a supernode is destined for a teal or yellow node of the same supernode. Also by Observation 144, no other token slides in $\vec{T}$. Thus, for each vertex $u \in V(D)$ the tokens on the orange nodes of supernode u must first slide to a purple connected side rail node (as all the nodes in T are inside the vertex-gadgets). From Observation 146, we can assume that the tokens on these orange nodes slide first to their purple rail neighbors using one token slide each.

For each vertex $u \in V(D)$, we analyze the token slides needed to fill the yellow nodes of supernode u . Consider a single token on a brown node of supernode u that is closest to the transit node for u . If this token is destined for the yellow node of supernode u that is token-free and farthest from the bottom-most purple connected side rail node of sub-ladder 1 of H_u , then this single token will require $4\Sigma(t+1) - \delta(u) + 4$ token slides.

Alternatively, consider the following token movement pattern to fill a yellow node of supernode u . A token on a brown node of supernode u that contains a token and is closest to the transit node for u is destined for one teal node y of supernode u that does not contain a token. The token on the purple connected side rail node x (with y as a rail neighbor), which came from the orange node of supernode u adjacent to x , is destined for the yellow node of supernode u that is token-free and is farthest from the bottom-most purple-colored node of sub-ladder 1 of H_u . This movement pattern requires $4\Sigma(t+1) - \delta(u) + 1$ token slides to get one token to a yellow node. Note that this is the minimum number of token slides possible to get a token to a yellow node of supernode u from one of the other (orange or brown) nodes of supernode u (and these are the only nodes from which a token can slide into a yellow node of supernode u as noted above as a consequence of Lemma 150 and Observation 144), and this minimum can only be achieved in this manner.

Given that $b = \Delta + \sum_{u \in V(D)} \delta(u) \cdot (4\Sigma(t+1) - \delta(u))$ and for each $u \in V(D)$, $\delta(u)$ tokens must slide into the yellow nodes of supernode u incurring each at minimum $4\Sigma(t+1) - \delta(u) + 1$ token slides, we see that tokens that end up on the yellow nodes must slide using the described token movement pattern. Consequently, for each vertex $u \in V(D)$, the number of orange nodes of supernode u is $\delta(u)$. Additionally, since T satisfies conditions (B4) and (B5), for each arc-gadget H_{uv} corresponding to an arc $uv \in A(D)$ incident to u and v , the orange token-bearing nodes of exactly one sub-ladder are token-free in T . Thus, the orange nodes of a supernode are made up of the appropriate token-bearing nodes corresponding to exactly one selected integer $i \in [t]$ for each arc-gadget representing an arc incident to u . Hence, for each arc, exactly one index $i \in [t]$ is selected and its tuple supplies exactly its first value to the tail vertex and its second value to the head vertex, and this supply equals the demand of the vertices. Thus, (D, δ, λ) is a yes-instance of PLANAR ARC SUPPLY. $\square$

Chapter 8

Conclusion and Directions For Future Work

This thesis has explored the classical and parameterized complexity landscape of combinatorial transformation problems, with a particular focus on solution discovery and reconfiguration problems.

Investigation of the parameterized complexity of our reconfiguration problems highlighted a dichotomy between labeled and unlabeled variants of the problem, rather than between directed and undirected ones, when the parameter is the budget b . Specifically, we demonstrate that the problems become fixed-parameter tractable with unlabeled tokens but remain intractable for labeled ones (that is, those that represent distinguishable objects we wish to slide). This mirrors the behavior of the same problem with a transformation step that is a single token slide; this problem is polynomial-time solvable with unlabeled tokens even with graph orientation but becomes **NP**-hard with labeled tokens on undirected graphs [101], which therefore implies intractability for the directed case. For the parameter number of tokens, we obtain fixed-parameter tractability results only for the case of unlabeled tokens; the status for labeled tokens, on both undirected and directed graphs, remains open. Likewise, while we proved parameterized hardness for the parameter b in the labeled setting, a natural next step is to sharpen this hardness by asking whether these variants, starting from the directed one, are hard for higher levels of the **W**-hierarchy. On preprocessing, not all our current tractability results give kernel guarantees. It is worth testing whether parameter b admits different preprocessing behavior with unlabeled tokens on directed versus undirected graphs, for example by establishing polynomial kernels or ruling them out. Any such strengthening could potentially clarify whether there is a directed-undirected complexity gap.

For solution discovery problems which are a relatively new area, several interesting questions remain unanswered. Combining the results of our work and that of previous work, we first notice that movement rather than the property’s computational complexity is what drives classical hardness. Namely, MATCHING-DISCOVERY, VERTEX/EDGE CUT-DISCOVERY, and SHORTEST PATH-DISCOVERY, with properties that are polynomial-time solvable, are NP-hard. However, there also exist solution discovery problems that are in P. In particular, Grobler et al. [103] have shown that when the property is to form a spanning tree in the graph, then the solution discovery variant, named SPANNING TREE DISCOVERY is polynomial-time solvable through a similar technique to the one we adapt for VERTEX CUT-DISCOVERY when parameterized by k . Second, the budget parameter b alone has proven too weak for all solution discovery problems on general graphs, where the algorithm leverages the problem’s inherent distance-based structure (that is, the property that there must exist exactly one token per level in this structure that exists between the vertices s and t). In fact, unlike VERTEX COVER-DISCOVERY, each of INDEPENDENT SET-DISCOVERY and DOMINATING SET-DISCOVERY remain hard even when we combine the budget b and the number of tokens k in a parameter. Third, when parameterized by k , our and previous work results on solution discovery problems with polynomial-time solvable properties indicate tractability for MATCHING-DISCOVERY, VERTEX/EDGE CUT-DISCOVERY, and SHORTEST PATH-DISCOVERY, but intractability when the tokens must simply form a grid [63]. These observations collectively raise fundamental questions: “What structural properties determine whether a polynomial-time solvable problem yields a tractable discovery variant?”, “Can we characterize conditions under which parameter b suffices for fixed-parameter tractability even on dense graphs?”, “Can we completely characterize when the parameter k guarantees tractability for discovery variants of polynomial-time solvable properties?”. Answering these questions would provide a unified theory of solution discovery complexity.

On the topic of unified theories, our results reveal striking complexity thresholds. For properties expressible in first-order logic (FO), solution discovery encounters meta-hardness barriers at surprisingly large structural parameters such as twin cover, modulators to stars, and modulators to paths numbers. This directly implies meta-hardness for the more expressive monadic second-order logic (MSO) with the same parameters. When we consider properties expressible in MSO, then neighborhood diversity suffices for fixed-parameter meta-tractability, but bandwidth does not. Combining these results first suggests a fundamental tension between logical expressivity and discovery complexity. While deciding whether a graph satisfies such logical properties can be done efficiently given these structural parameters, the setting in discovery problems appears to interact with logical quantification in ways that rapidly escalate complexity. Second, it highlights the gaps that

remain in our investigation of the meta-tractability of these problems with respect to structural parameters. Thus we ask “Do solution discovery variants of properties expressible in FO become tractable under bandwidth or the even smaller parameter bisection width?”, “Do solution discovery variants of properties expressible in FO become tractable under the parameter maximum leaf number?”. We also observe that the properties for which we establish hardness tend to exhibit a numeric-constraint-checking character, even when the source problem in the reduction does not, such as the MULTICOLORED CLIQUE problem in the FPT-reductions for the modulator to stars and modulator to paths parameters. This observation can point toward a deeper characterization: “Do solution discovery problems of properties defined by numeric constraints resist fixed-parameter tractability?”. More generally, “Can a precise characterization of the properties that give rise to $W[1]$ -hard solution discovery problems be established?”.

As a further point, while our kernelization results vary across problems, they signal that preprocessing success depends heavily on both the specific problem formulation and, naturally, the graph class. For example, MATCHING-DISCOVERY admits a polynomial kernel with respect to the parameter k with a preprocessing algorithm that takes advantage of the locality of the matching constraint, whereas VERTEX/EDGE CUT-DISCOVERY and SHORTEST PATH-DISCOVERY with no such local constraints do not (under standard assumptions). This raises fundamental questions about which problem characteristics, such as local checkability, enable effective preprocessing with parameter k . Our results also prove lower bounds excluding polynomial kernels for parameter b +pathwidth for each of INDEPENDENT SET-DISCOVERY, VERTEX COVER-DISCOVERY, and DOMINATING SET-DISCOVERY, naturally prompting investigation of larger parameters: “Does b +treedepth or b +vertex cover number, which dominate pathwidth, restore kernelization?”. Moreover, fine-grained kernelization complexity remains unexplored. For instance, “While MATCHING-DISCOVERY admits a polynomial kernel, can we achieve compressed instances of smaller size than $\mathcal{O}(k^5)$, or is the latter proven bound optimal?”. Beyond that, our kernelization lower bound for SHORTEST PATH-DISCOVERY extends naturally from the classical hardness reduction, suggesting that it would be worthwhile to develop a general framework characterizing when such extensions are possible and under what conditions they can be systematically applied.

Lastly, we note that several intriguing questions arise from our case-by-case analysis of solution discovery problems whose underlying properties are polynomial-time solvable. In nearly all such hardness results, we were able to establish (or could plausibly establish) that the reductions hold even under bounded degeneracy. This recurring phenomenon motivates the question of whether a general framework can be developed that systematically extends hardness results for solution discovery problems to classes of graphs of bounded degeneracy.

Additionally, since the fixed-parameter tractable algorithm for VERTEX-CUT DISCOVERY operates in polynomial time for every fixed k , it is natural to only ask whether a simpler or more direct algorithm can be designed for the special case $k = 1$, corresponding to identifying the nearest bridge cut.

References

- [1] H. L. Abbott, Denis Hanson, and Norbert Sauer. Intersection theorems for systems of sets. *J. Comb. Theory A*, 12(3):381–389, 1972.
- [2] Mikkel Abrahamsen, Kevin Buchin, Maike Buchin, Linda Kleist, Maarten Löffler, Lena Schlipf, André Schulz, and Jack Stade. Reconfiguration of unit squares and disks: PSPACE-hardness in simple settings. In *41st International Symposium on Computational Geometry (SoCG 2025)*, volume 332 of *LIPIcs*, pages 1:1–1:18, 2025.
- [3] Kira V. Adaricheva, Heather C. Smith Blake, Chassidy Bozeman, Nancy E. Clarke, Ruth Haas, Margaret-Ellen Messinger, and Karen Seyffarth. Hamilton paths in dominating graphs of trees and cycles. *Graphs Comb.*, 38(6):174, 2022.
- [4] Sara Ahmadian, Zachary Friggstad, and Chaitanya Swamy. Local-search based approximation algorithms for mobile facility location problems. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1607–1621. SIAM, 2013.
- [5] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008.
- [6] Noga Alon and Shai Gutner. Balanced families of perfect hash functions and their applications. *ACM Trans. Algorithms*, 6(3):54:1–54:12, 2010.
- [7] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

- [8] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 56–68. Springer, 2015.
- [9] Amitai Armon, Iftah Gamzu, and Danny Segev. Mobile facility location: combinatorial filtering via weighted occupancy. *J. Comb. Optim.*, 28(2):358–375, 2014.
- [10] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.
- [11] David Avis and Duc A. Hoang. A note on acyclic token sliding reconfiguration graphs of independent sets. *Ars Comb.*, 159(1):133–154, 2024.
- [12] Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- [13] Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. *Algorithmica*, 83(9):2914–2951, 2021.
- [14] Valentin Bartier, Nicolas Bousquet, Jihad Hanna, Amer E. Mouawad, and Sebastian Siebertz. Token sliding on graphs of girth five. *Algorithmica*, 86(2):638–655, 2024.
- [15] Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *J. Comput. Syst. Sci.*, 136:220–248, 2023.
- [16] Andreas Bärtschi, Evangelos Bampas, Jérémie Chalopin, Shantanu Das, Christina Karousatou, and Matús Mihalák. Near-gathering of energy-constrained mobile agents. *Theor. Comput. Sci.*, 849:35–46, 2021.
- [17] Rémy Belmonte, Tesshu Hanaka, Michael Lampis, Hirotaka Ono, and Yota Otachi. Independent set reconfiguration parameterized by modular-width. *Algorithmica*, 82(9):2586–2605, 2020.

- [18] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.
- [19] Piotr Berman, Erik D. Demaine, and Morteza Zadimoghaddam. $O(1)$ -approximations for maximum movement problems. In Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, volume 6845 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 2011.
- [20] Aaron Bernstein, Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Deterministic dynamic maximal matching in sublinear update time. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 132–143. ACM, 2025.
- [21] Marzio De Biasi and Juho Lauri. On the complexity of restoring corrupted colorings. *J. Comb. Optim.*, 37(4):1150–1169, 2019.
- [22] Davide Bilò. New algorithms for steiner tree reoptimization. *Algorithmica*, 86(8):2652–2675, 2024.
- [23] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Exact and approximate algorithms for movement problems on (special classes of) graphs. *Theor. Comput. Sci.*, 652:86–101, 2016.
- [24] Ahmad Biniaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. *Discret. Math. Theor. Comput. Sci.*, 24(2), 2022.
- [25] Alexandre Blanché, Haruka Mizuta, Paul Ouvrard, and Akira Suzuki. Decremental optimization of dominating sets under the reconfiguration framework. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, volume 12126 of *Lecture Notes in Computer Science*, pages 69–82. Springer, 2020.
- [26] Hans-Joachim Böckenhauer, Elisabet Burjons, Martin Raszyk, and Peter Rossmanith. Reoptimization of parameterized problems. *Acta Informatica*, 59(4):427–450, 2022.

- [27] Pierre Le Bodic and Edward Lam. Optimal unlabeled pebble motion on trees. In Ariel Felner and Jiaoyang Li, editors, *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, pages 218–222. AAAI Press, 2024.
- [28] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- [29] Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022.
- [30] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Michael T. Hallett, and Harold T. Wareham. Parameterized complexity analysis in computational biology. *Comput. Appl. Biosci.*, 11(1):49–57, 1995.
- [31] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.
- [32] Hans L. Bodlaender, Michael R. Fellows, and Michael T. Hallett. Beyond np-completeness for problems of bounded width: hardness for the W hierarchy. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 449–458. ACM, 1994.
- [33] Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. Xnlp-completeness for parameterized problems on graphs with a linear structure. *Algorithmica*, 87(4):465–506, 2025.
- [34] Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. *Inf. Comput.*, 300:105195, 2024.
- [35] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014.

- [36] Hans L. Bodlaender, Daniel Lokshantov, and Eelko Penninkx. Planar capacitated dominating set is $W[1]$ -hard. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 50–60. Springer, 2009.
- [37] Édouard Bonnet, Tillmann Miltzow, and Pawel Rzazewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.
- [38] Nicolas Bousquet, Quentin Deschamps, Arnaud Mary, Amer E. Mouawad, and Théo Pierron. The tape reconfiguration problem and its consequences for dominating set reconfiguration. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms, ESA 2025, September 15-17, 2025, Warsaw, Poland*, volume 351 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [39] Nicolas Bousquet, Kshitij Gajjar, Abhiruk Lahiri, and Amer E. Mouawad. Parameterized shortest path reconfiguration. In Édouard Bonnet and Pawel Rzazewski, editors, *19th International Symposium on Parameterized and Exact Computation, IPEC 2024, September 4-6, 2024, Royal Holloway, University of London, Egham, United Kingdom*, volume 321 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [40] Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of reconfiguration problems. *Comput. Sci. Rev.*, 53:100663, 2024.
- [41] Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of reconfiguration problems. *Comput. Sci. Rev.*, 53:100663, 2024.
- [42] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 1999.
- [43] Robert Brederick, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *CoRR*, abs/1407.2143, 2014. preprint.
- [44] Robert Brederick, Jiehua Chen, Piotr Faliszewski, André Nichterlein, and Rolf Niedermeier. Prices matter for the parameterized complexity of shift bribery. In Carla E.

- Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1398–1404. AAAI Press, 2014.
- [45] Jop Briët, Harry Buhrman, Debbie W. Leung, Teresa Piovesan, and Florian Speelman. Round elimination in exact communication complexity. In Salman Beigi and Robert König, editors, *10th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2015, May 20-22, 2015, Brussels, Belgium*, volume 44 of *LIPICs*, pages 206–225. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
 - [46] Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: An overview. *Algorithms*, 12(12):256, 2019.
 - [47] Benjamin A. Burton and Rodney G. Downey. Courcelle’s theorem for triangulations. *J. Comb. Theory A*, 146:264–294, 2017.
 - [48] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Log.*, 84(1):119–138, 1997.
 - [49] Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008.
 - [50] Barry Cimring, Remy El Sabeh, Marc Bacvanski, Stephanie Maaz, Izzat El Hajj, Naomi Nishimura, Amer E Mouawad, and Alexandre Cooper. Efficient algorithms to solve atom reconfiguration problems. i. redistribution-reconfiguration algorithm. *Physical Review A*, 108(2):023107, 2023.
 - [51] Vincent Cohen-Addad, Anupam Gupta, Lunjia Hu, Hoon Oh, and David Saulpic. An improved local search algorithm for k-median. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1556–1612. SIAM, 2022.
 - [52] Alexandre Cooper, Stephanie Maaz, Amer E. Mouawad, and Naomi Nishimura. Parameterized complexity of reconfiguration of atoms. In Petra Mutzel, Md. Saidur Rahman, and Slamin, editors, *WALCOM: Algorithms and Computation - 16th International Conference and Workshops, WALCOM 2022, Jember, Indonesia, March 24-26, 2022, Proceedings*, volume 13174 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2022.

- [53] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- [54] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
- [55] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.
- [56] Gruia Cuarescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM J. Discret. Math.*, 22(1):124–138, 2008.
- [57] Luís Felipe I. Cunha, Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Computing distances on graph associahedra is fixed-parameter tractable. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, volume 334 of *LIPIcs*, pages 63:1–63:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [58] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- [59] Ye Dai, Shilong He, Xinlei Nie, Xukun Rui, Shikun Li, and Sai He. Research on reconfiguration strategies for self-reconfiguring modular robots: A review. *Journal of Intelligent & Robotic Systems*, 110:47, 2024.
- [60] Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.
- [61] Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015.
- [62] Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Minimizing movement. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the*

Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007, pages 258–267. SIAM, 2007.

- [63] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. Minimizing movement: Fixed-parameter tractability. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 718–729. Springer, 2009.
- [64] M. H. Devoret, A. Wallraff, and J. M. Martinis. Superconducting Qubits: A Short Review. *arXiv e-prints*, November 2004. preprint.
- [65] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [66] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.
- [67] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for $W[1]$. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.
- [68] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
- [69] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
- [70] Jan Dreier, Ioannis Eleftheriadis, Nikolas Mählmann, Rose McCarty, Michal Pilipczuk, and Szymon Torunczyk. First-order model checking on monadically stable graph classes. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 21–30. IEEE, 2024.
- [71] Jan Dreier, Nikolas Mählmann, and Sebastian Siebertz. First-order model checking on structurally sparse graph classes. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 567–580. ACM, 2023.
- [72] Stuart E. Dreyfus and Robert A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

- [73] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.
- [74] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 1995.
- [75] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.
- [76] Remy El Sabeh, Jessica Bohm, Zhiqian Ding, Stephanie Maaz, Naomi Nishimura, Izzat El Hajj, Amer E Mouawad, and Alexandre Cooper. Efficient algorithms to solve atom reconfiguration problems. ii. assignment-rerouting-ordering algorithm. *Physical Review A*, 108(2):023108, 2023.
- [77] Element Logic. What is autostore? <https://www.elementlogic.net/insights/what-is-autostore/>, 2024. Accessed: 2024.
- [78] Kazim Selim Engin and Volkan Isler. Establishing fault-tolerant connectivity of mobile robot networks. *IEEE Trans. Control. Netw. Syst.*, 8(2):667–677, 2021.
- [79] P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 01 1960.
- [80] Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPICs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [81] Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive algorithms for domination and independence. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [82] Michael R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and*

Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings, volume 4169 of *Lecture Notes in Computer Science*, pages 276–277. Springer, 2006.

- [83] Michael R. Fellows, Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Frances A. Rosamond, Daniel Schmand, and Sebastian Siebertz. On solution discovery via reconfiguration. In Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu, editors, *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)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 700–707. IOS Press, 2023.
- [84] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.
- [85] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- [86] Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Saket Saurabh, and Meirav Zehavi. Kernelization for spreading points. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 48:1–48:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [87] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- [88] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.
- [89] Zachary Friggstad and Mohammad R. Salavatipour. Minimizing movement in mobile facility location problems. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, Philadelphia, PA, USA, October 25-28, 2008*, pages 357–366. IEEE Computer Society, 2008.
- [90] Vincent Froese, Christoph Hertrich, and Rolf Niedermeier. The computational complexity of relu network training parameterized by data dimensionality. *J. Artif. Intell. Res.*, 74:1775–1790, 2022.

- [91] Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011.
- [92] Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. Meta-kernelization with structural parameters. *J. Comput. Syst. Sci.*, 82(2):333–346, 2016.
- [93] M. R. Garey, David S. Johnson, and Robert Endre Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.
- [94] Valentin Garnero, Konstanty Junosza-Szaniawski, Mathieu Liedloff, Pedro Montealegre, and Pawel Rzazewski. Fixing improper colorings of graphs. *Theor. Comput. Sci.*, 711:66–78, 2018.
- [95] Jaroslav Garvardt, Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. Parameterized local search for max c-cut. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 5586–5594. ijcai.org, 2023.
- [96] Mehrdad Ghadiri and Sina Yazdanbod. Minimizing the total movement for movement to independence problem on a line. In Thomas C. Shermer, editor, *Proceedings of the 28th Canadian Conference on Computational Geometry, CCCG 2016, August 3-5, 2016, Simon Fraser University, Vancouver, British Columbia, Canada*, pages 15–20. Simon Fraser University, Vancouver, British Columbia, Canada, 2016.
- [97] Abdolhamid Ghodselahi and Fabian Kuhn. Serving online requests with mobile servers. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 740–751. Springer, 2015.
- [98] Abdolhamid Ghodselahi and Fabian Kuhn. Toward online mobile facility location on general metrics. *Theory Comput. Syst.*, 67(6):1268–1306, 2023.
- [99] Tatsuya Gima, Takehiro Ito, Yasuaki Kobayashi, and Yota Otachi. Algorithmic meta-theorems for combinatorial reconfiguration revisited. *Algorithmica*, 86(11):3395–3424, 2024.

- [100] Andrew V. Goldberg and Robert Endre Tarjan. Solving minimum-cost flow problems by successive approximation. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 7–18. ACM, 1987.
- [101] Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 2011.
- [102] Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures with bounded treewidth. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 165–174. ACM, 2007.
- [103] Mario Grobler, Stephanie Maaz, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, and Sebastian Siebertz. Solution discovery via reconfiguration for problems in P. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 76:1–76:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [104] Mario Grobler, Stephanie Maaz, Amer E. Mouawad, Naomi Nishimura, Vijayaragunathan Ramamoorthi, and Sebastian Siebertz. Kernelization complexity of solution discovery problems. In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia*, volume 322 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [105] Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004.
- [106] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017.
- [107] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

- [108] Russell D. Halper, S. Raghavan, and Mustafa Sahin. Local search heuristics for the mobile facility location problem. *Comput. Oper. Res.*, 62:210–223, 2015.
- [109] Tesshu Hanaka, Yuni Iwamasa, Yasuaki Kobayashi, Yuto Okada, and Rin Saito. Basis sequence reconfiguration in the union of matroids. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, volume 322 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [110] Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.
- [111] Shuichi Hirahara and Naoto Ohsaka. Optimal pspace-hardness of approximating set cover reconfiguration. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *LIPIcs*, pages 85:1–85:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [112] Shuichi Hirahara and Naoto Ohsaka. Probabilistically checkable reconfiguration proofs and inapproximability of reconfiguration problems. In *56th Annual ACM Symposium on Theory of Computing (STOC 2024)*, pages 1435–1445, 2024.
- [113] Duc A. Hoang, Amanj Khorramian, and Ryuhei Uehara. Shortest reconfiguration sequence for sliding tokens on spiders. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2019.
- [114] Duc A. Hoang and Ryuhei Uehara. Sliding tokens on a cactus. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, volume 64 of *LIPIcs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [115] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.
- [116] Jacob Holm, Wojciech Nadara, Eva Rotenberg, and Marek Sokolowski. Fully dynamic biconnectivity in $\tilde{O}(\log^2 n)$ time. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 156–165. ACM, 2025.
- [117] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

- [118] John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “warehouseman’s problem”. *The International Journal of Robotics Research*, 3(4):76–88, 1984.
- [119] Kai Hoppmann-Baum. On the complexity of computing maximum and minimum min-cost-flows. *Networks*, 79(2):236–248, 2022.
- [120] Falk Hüffner, Christian Komusiewicz, Rolf Niedermeier, and Sebastian Wernicke. Parameterized algorithmics for finding exact solutions of NP-hard biological problems. In *Methods in Molecular Biology*, volume 1526, pages 363–402. Springer, 2017.
- [121] Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008.
- [122] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
- [123] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.
- [124] Takehiro Ito, Haruka Mizuta, Naomi Nishimura, and Akira Suzuki. Incremental optimization of independent sets under the reconfiguration framework. *J. Comb. Optim.*, 43(5):1264–1279, 2022.
- [125] Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012.
- [126] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
- [127] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013.
- [128] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle’s theorem - A game-theoretic approach. *Discret. Optim.*, 8(4):568–594, 2011.

- [129] Dusan Knop, Martin Koutecký, Tomáš Masarík, and Tomáš Toufar. Simplified algorithmic metatheorems beyond MSO: treewidth and neighborhood diversity. *Log. Methods Comput. Sci.*, 15(4), 2019.
- [130] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, David I Schuster, Johannes Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. Charge-insensitive qubit design derived from the cooper pair box. *Physical Review A*, 76(4):042319, 2007.
- [131] Petr Kolman, Martin Koutecký, and Hans Raj Tiwary. Extension complexity, MSO logic, and treewidth. *Discret. Math. Theor. Comput. Sci.*, 22(4), 2020.
- [132] Daniel Kornhauser, Gary L. Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 241–250, 1984.
- [133] Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Trans. Algorithms*, 15(2):24:1–24:19, 2019.
- [134] Jan Matyáš Kristan and Jakub Svoboda. Shortest dominating set reconfiguration under token sliding. In Henning Fernau and Klaus Jansen, editors, *Fundamentals of Computation Theory - 24th International Symposium, FCT 2023, Trier, Germany, September 18-21, 2023, Proceedings*, volume 14292 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2023.
- [135] Jan Matyáš Kristan and Jakub Svoboda. Reconfiguration using generalized token jumping. In Shin-ichi Nakano and Mingyu Xiao, editors, *WALCOM: Algorithms and Computation - 19th International Conference and Workshops on Algorithms and Computation, WALCOM 2025, Chengdu, China, February 28 - March 2, 2025, Proceedings*, volume 15411 of *Lecture Notes in Computer Science*, pages 244–265. Springer, 2025.
- [136] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.
- [137] Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theor. Comput. Sci.*, 524:27–33, 2014.

- [138] Shimin Li and Haitao Wang. Algorithms for minimizing the movements of spreading points in linear domains. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*. Queen’s University, Ontario, Canada, 2015.
- [139] Guanqi Liang, Di Wu, Yuxiao Tu, and Tin Lun Lam. Decoding modular reconfigurable robots: A survey on mechanisms and design. *Int. J. Robotics Res.*, 44(5):740–767, 2025.
- [140] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
- [141] Josef Malík, Ondrej Suchý, and Tomás Valla. Efficient implementation of color coding algorithm for subgraph isomorphism problem. In Ilias S. Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA² 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers*, volume 11544 of *Lecture Notes in Computer Science*, pages 283–299. Springer, 2019.
- [142] Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010.
- [143] Dániel Marx, Barry O’Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, 2013.
- [144] Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999.
- [145] Timo Camillo Merkl and Reinhard Pichler. Parameterised complexity of consistent query answering via graph representations. *CoRR*, abs/2412.08324, 2024. preprint.
- [146] Neeldhara Misra, Frances A. Rosamond, and Meirav Zehavi. Special issue ”new frontiers in parameterized complexity and algorithms”: Foreward by the guest editors. *Algorithms*, 13(9):236, 2020.
- [147] Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.
- [148] Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC*

- 2014, Wroclaw, Poland, September 10-12, 2014. *Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2014.
- [149] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995.
 - [150] Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.
 - [151] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discret. Appl. Math.*, 152(1-3):229–245, 2005.
 - [152] Sebastian Ordyniak, Giacomo Paesani, Mateusz Rychlicki, and Stefan Szeider. Explaining decisions in ML models: A parameterized complexity analysis. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024.
 - [153] Alon Orlitsky. Worst-case interactive communication I: two messages are almost optimal. *IEEE Trans. Inf. Theory*, 36(5):1111–1126, 1990.
 - [154] Lucie Pansart, Hadrien Cambazard, and Nicolas Catusse. New randomized strategies for the color coding algorithm. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 466–473. IOS Press, 2020.
 - [155] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
 - [156] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.
 - [157] Thomy Phan, Benran Zhang, Shao-Hung Chan, and Sven Koenig. Anytime multi-agent path finding with an adaptive delay-based heuristic. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-25)*, 2025.

- [158] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 694–705. Springer, 2009.
- [159] Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. On the number of types in sparse graphs. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 799–808. ACM, 2018.
- [160] S. Raghavan, Mustafa Sahin, and F. Sibel Salman. The capacitated mobile facility location problem. *Eur. J. Oper. Res.*, 277(2):507–520, 2019.
- [161] S. Raghavan and Rui Zhang. Rapid influence maximization on social networks: The positive influence dominating set problem. *INFORMS J. Comput.*, 34(3):1345–1365, 2022.
- [162] Lizeth C. Riascos-Álvarez, Merve Bodur, and Dionne M. Aleman. A branch-and-price algorithm enhanced by decision diagrams for the kidney exchange problem. *Manuf. Serv. Oper. Manag.*, 26(2):485–499, 2024.
- [163] Marion Roman, Barbara Catherine, and Nathalie Villa-Vialaneix. Feature selection for kernel methods in systems biology. *NAR Genomics and Bioinformatics*, 4(1):lqac014, 2022.
- [164] Norbert Sauer. On the density of families of sets. *J. Comb. Theory A*, 13(1):145–147, 1972.
- [165] Kai-Niklas Schymik, Vincent Lienhard, Daniel Barredo, Pascal Scholl, Hannah Williams, Antoine Browaeys, and Thierry Lahaye. Enhanced atom-by-atom assembly of arbitrary tweezer arrays. *Physical Review A*, 102(6):063107, 2020.
- [166] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.
- [167] Sebastian Siebertz. Greedy domination on biclique-free graphs. *Inf. Process. Lett.*, 145:64–67, 2019.
- [168] Sebastian Siebertz. Personal communication, 2025.

- [169] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. Qubit allocation. In Jens Knoop, Markus Schordan, Teresa Johnson, and Michael F. P. O’Boyle, editors, *Proceedings of the 2018 International Symposium on Code Generation and Optimization, CGO 2018, Vösendorf / Vienna, Austria, February 24-28, 2018*, pages 113–125. ACM, 2018.
- [170] Sotefin Bharat. Automated puzzle car parking systems: Innovative solutions. <https://www.sotefinbharat.com/products/puzzle-car-parking-system/>, 2025. Accessed: May 9, 2025.
- [171] MIT-NASA Space Robots Team, Josh Brunner, Kenneth C. Cheung, Erik D. Demaine, Jenny Diomidova, Christine Gregg, Della H. Hendrickson, and Irina Kostitsyna. Reconfiguration algorithms for cubic modular robots with realistic movement constraints. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- [172] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Track A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008.
- [173] Onur Tekdas, Yokesh Kumar, Volkan Isler, and Ravi Janardan. *Building a Communication Bridge with Mobile Hubs*, page 179–190. Springer-Verlag, Berlin, Heidelberg, 2009.
- [174] Tom Tseng, Laxman Dhulipala, and Julian Shun. Parallel batch-dynamic minimum spanning forest and the efficiency of dynamic agglomerative graph clustering. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA ’22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 233–245. ACM, 2022.
- [175] Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark J. Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013.

- [176] Vijay V. Vazirani. A proof of the MV matching algorithm. *CoRR*, abs/2012.03582, 2020.
- [177] Hao Wang, Zhiying Huang, Yanqing Liang, Qingxi Zhang, Shaoxiong Hu, Liye Cui, and Xiangyun An. Dynamic evolution of urban infrastructure resilience and its spatial spillover effects: An empirical study from china. *PLOS ONE*, 18(3):1–20, 03 2023.
- [178] Xuefeng Wang, Eric P. Xing, and Daniel J. Schaid. Kernel methods for large-scale genomic data analysis. *Briefings Bioinform.*, 16(2):183–192, 2015.
- [179] Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice hall, 2001.
- [180] Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018.
- [181] Takeshi Yamada and Ryuhei Uehara. Shortest reconfiguration of sliding tokens on a caterpillar. In Mohammad Kaykobad and Rossella Petreschi, editors, *WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings*, volume 9627 of *Lecture Notes in Computer Science*, pages 236–248. Springer, 2016.
- [182] Yusuke Yanagisawa, Akira Suzuki, Yuma Tamura, and Xiao Zhou. Decremental optimization of vertex-coloring under the reconfiguration framework. In *COCOON*, volume 13025 of *LNCS*, pages 355–366. Springer, 2021.
- [183] Yusuke Yanagisawa, Akira Suzuki, Yuma Tamura, and Xiao Zhou. Parameterized complexity of optimizing list vertex-coloring through reconfiguration. In *IWOCA*, volume 13889 of *LNCS*, pages 271–282. Springer, 2023.
- [184] Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.
- [185] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, pages 1443–1449. AAAI Press, 2013.
- [186] Yu Zhang, Shengzhi Wang, Chanjuan Liu, and Enqiang Zhu. TIVC: an efficient local search algorithm for minimum vertex cover in large graphs. *Sensors*, 23(18):7831, 2023.

APPENDICES

Appendix A

Formulas for Solution Discovery Hardness Meta Theorems

A.1 Formulas for FO-Discovery for Modulator to Stars and Modulator to Paths Numbers

A.1.1 Formulas for FO-Discovery for Modulator to Stars Num- ber

We write here a formula $\phi(X)$ for the conditions of the proof of Theorem 128. We assume the following predicate, $\text{Col}(x) := \bigvee_{i \in [5]} C_i(x)$, which holds if x is a colored node in $(H, \mathcal{C})$ of Section 7.2.1 is available. Given this, we write

S1. No colored node contains a token.

$$\forall x \in V(H) : (\text{Col}(x) \rightarrow \neg X(x)),$$

S2. For each teal (C_2) (vertex block) node, there exists exactly one adjacent (purple) (C_1) (vertex root) node such that all the uncolored neighbors (leaves) of this (purple) node do not contain tokens. Moreover, all the other (purple) (vertex root) nodes adjacent to this teal node have tokens on all the uncolored nodes (leaves) in their

neighborhoods.

$$\begin{aligned} \forall x \in V(H) : \left(C_2(x) \rightarrow \left[\exists r \in V(H) : \left(E(x, r) \wedge \forall z \in V(H) : \left[(E(r, z) \wedge \neg \text{Col}(z)) \rightarrow \neg X(z) \right] \right) \wedge \forall q \in V(H) : \left((E(x, q) \wedge (q \neq r)) \rightarrow \right. \right. \right. \\ \left. \left. \left[\forall z \in V(H) : (E(q, z) \wedge \neg \text{Col}(z)) \rightarrow X(z) \right] \right) \right] \right), \end{aligned}$$

- S3. For each gray (C_4) (edge block) node, there exists exactly one adjacent (black) (C_3) (edge root) node such that all the uncolored neighbors (leaves) of this (black) node contain tokens. Moreover, all the other (edge root) nodes adjacent to this gray node are such that all the uncolored nodes (leaves) in their neighborhoods are token-free.

$$\begin{aligned} \forall x \in V(H) : \left(C_4(x) \rightarrow \left[\exists r \in V(H) : \left(E(x, r) \wedge \forall z \in V(H) : \left[(E(r, z) \wedge \neg \text{Col}(z)) \rightarrow X(z) \right] \right) \wedge \forall q \in V(H) : \left((E(x, q) \wedge (q \neq r)) \rightarrow \right. \right. \right. \\ \left. \left. \left[\forall z \in V(H) : (E(q, z) \wedge \neg \text{Col}(z)) \rightarrow \neg X(z) \right] \right) \right] \right), \end{aligned}$$

and finally set $\phi(X) = S1(X) \wedge S2(X) \wedge S3(X)$.

A.1.2 Formulas for FO-Discovery for Modulator to Paths Number

We write here a formula $\phi(X)$ for the conditions of the proof of Theorem 133. We assume that the following predicates are available:

- x is a colored node in $(H, \mathcal{C})$ of Section 7.2.2 if and only if this holds:

$$\text{Col}(x) := \bigvee_{i \in [5]} C_i(x).$$

- z is a node in the distance-2 neighborhood of x and is reachable from x through a node that is not a teal (C_2) edge block node, or a gray (C_4) edge block node, or a green (C_5) connector node, if and only if this holds:

$$\text{Dist2NoC245}(x, z) := \exists y : \text{Edge}(x, y) \wedge \text{Edge}(y, z) \wedge \neg C_2(y) \wedge \neg C_4(y) \wedge \neg C_5(y).$$

Given this, we write

P1. No colored node contains a token.

$$\forall x \in V(H) : (\text{Col}(x) \rightarrow \neg X(x)),$$

P2. For each teal (C_2) (vertex block) node, there exists at least one adjacent uncolored node (in a path vertex-gadget) that is token-free.

$$\forall x \in V(H) : [C_2(x) \rightarrow \exists p \in V(H) : (E(x, p) \wedge \neg \text{Col}(p) \wedge \neg X(p))],$$

P3. For each gray (C_4) (edge block) node, there exists at least one adjacent uncolored node (in a path edge-gadget) that contains a token.

$$\forall x \in V(H) : [C_4(x) \rightarrow \exists p \in V(H) : (E(x, p) \wedge \neg \text{Col}(p) \wedge X(p))],$$

P4. Every uncolored node v (in a gadget) contains a token if and only if all uncolored nodes (in a gadget) at distance at most two from v in $H[V \setminus (C_2 \cup C_4 \cup C_5)]$ also contain tokens.

$$\begin{aligned} \forall x \in V(H) : & \left(\neg \text{Col}(x) \rightarrow [X(x) \rightarrow \forall z \in V(H) : (\text{Dist2NoC245}(x, z) \rightarrow \right. \\ & X(z))] \wedge [\neg X(x) \rightarrow \forall z \in V(H) : (\text{Dist2NoC245}(x, z) \rightarrow \\ & \left. \neg X(z))] \right], \end{aligned}$$

and set $\phi(X) = \text{P1}(X) \wedge \text{P2}(X) \wedge \text{P3}(X) \wedge \text{P4}(X)$.

A.2 Formulas for FO-Discovery for Twin Cover Number

We write here a formula $\phi(X)$ for the conditions of the proof of Theorem 138. We assume the following predicate, $\text{Col}(x) := \bigvee_{i \in [6]} C_i(x)$, which holds if x is a colored node in $(H, \mathcal{C})$ of Section 7.3 is available. Given this, we write

T1. Each yellow (C_3) (demand clique) node contains a token.

$$\forall x \in V(H) : (C_3(x) \rightarrow X(x)),$$

T2. The brown (C_6) (reservoir clique) nodes do not contain tokens.

$$\forall x \in V(H) : (C_6(x) \rightarrow \neg X(x)),$$

T3. For each pair of purple (C_1) (vertex) and black (C_2) (arc) nodes, exactly one adjacent (supply) clique containing at least one orange (C_4) (supply) node must be such that all its teal (C_5) (index) nodes contain tokens and all its orange (C_4) (supply) nodes do not. For all other adjacent (supply) cliques of this type, their orange (C_4) (supply) nodes contain tokens, and their teal (C_5) (index) nodes do not.

$$\begin{aligned} \forall x, y \in V(H) : & \left[(C_1(x) \wedge C_2(y)) \rightarrow \left(\exists p \in V(H) : \left[C_4(p) \wedge E(x, p) \wedge \right. \right. \right. \\ & \forall z \in V(H) : \left(\left[(C_4(z) \wedge E(p, z)) \rightarrow \neg X(z) \right] \wedge \left[(C_5(z) \wedge E(p, z)) \right. \right. \\ & \left. \left. \rightarrow X(z) \right] \right) \right] \wedge \forall q \in V(H) : \left[C_4(q) \wedge E(x, q) \wedge \neg E(p, q) \wedge (q \neq p) \right. \\ & \left. \rightarrow \forall z \in V(H) : \left(\left[(C_4(z) \wedge E(q, z)) \rightarrow X(z) \right] \wedge \left[(C_5(z) \wedge E(q, z)) \right. \right. \right. \\ & \left. \left. \left. \rightarrow \neg X(z) \right] \right) \right] \right], \end{aligned}$$

T4. The purple (C_1) (vertex) and black (C_2) (arc) nodes do not contain tokens.

$$\forall x \in V(H) : ((C_1(x) \vee C_2(x)) \rightarrow \neg X(x)),$$

and set $\phi(X) = T1(X) \wedge T2(X) \wedge T3(X) \wedge T4(X)$.

A.3 Formulas for MSO-Discovery for Bandwidth

We write here a formula $\phi(X)$ for the conditions of the proof of Theorem 141. We assume that the following predicates are available:

- x is a node of color gray (C_5) or black (C_6) or orange (C_7) if and only if this holds:

$$\text{Col}_{5,6,7}(x) := C_5(x) \vee C_6(x) \vee C_7(x).$$

- x and y are nodes at distance $k \in [5]$ if and only if this holds:

$$\text{Dist}_k(x, y) := \exists z_1, \dots, z_{k-1} (E(x, z_1) \wedge E(z_1, z_2) \wedge \dots \wedge E(z_{k-2}, z_{k-1}) \wedge E(z_{k-1}, y)).$$

- x and y lie in the same connected component formed of nodes of colors gray (C_5), black (C_6), and orange (C_7) only if and only if this holds:

$$\begin{aligned} \text{Con}_{5,6,7}(x, y) := & \text{Col}_{5,6,7}(x) \wedge \text{Col}_{5,6,7}(y) \wedge \neg \exists S \subseteq V(H) : \left(S(x) \wedge \neg S(y) \right. \\ & \wedge \forall z \in V(H) : (S(z) \rightarrow \text{Col}_{5,6,7}(z)) \wedge \forall u, v \in V(H) : [(S(u) \wedge \\ & \left. \text{Col}_{5,6,7}(u) \wedge E(u, v) \wedge \text{Col}_{5,6,7}(v)) \rightarrow S(v)] \right). \end{aligned}$$

(Intuitively, this formula states that x and y are connected if there is no subset S of nodes colored gray (C_5), black (C_6), or orange (C_7) that contains x but not y and forms a closed component under the edge relation. In other words, any path from x using only nodes of these three colors must be able to reach y .)

- x is a token-free orange (C_7) node that is at distance four to a black (C_6) node denoted y if and only if this holds:

$$\text{CTFC6}(x, y) := C_7(x) \wedge \neg X(x) \wedge C_6(y) \wedge \text{Dist}_4(x, y).$$

Given this, we write

- B1. The gray (C_5) (spacer), the purple (C_2) (non-demand connected side rail), and the brown (C_3) (reservoir) nodes do not contain tokens.

$$\forall x \in V(H) : [(C_5(x) \vee C_2(x) \vee C_3(x)) \rightarrow \neg X(x)],$$

- B2. Each yellow (C_1) (demand) node contains a token.

$$\forall x \in V(H) : (C_1(x) \rightarrow X(x)),$$

- B3. A teal (C_4) (isolated side rail) node contains a token if and only if an orange (C_7) (token-bearing) node is at distance five from it and does not contain a token.

$$\forall x \in V(H) : (C_4(x) \rightarrow [X(x) \leftrightarrow \exists y \in V(H) : (\text{Dist}_5(x, y) \wedge C_7(y) \wedge \neg X(y))]),$$

- B4. If an orange (C_7) (token-bearing) node contains a token, then the orange (C_7) (token-bearing) nodes at distance four from it also do. Otherwise, they do not:

$$\forall x \in V(H) : \left(C_7(x) \rightarrow \forall y \in V(H) : \left[(\text{Dist}_4(x, y) \wedge C_7(y)) \rightarrow (X(x) \leftrightarrow X(y)) \right] \right),$$

- B5. For each connected component formed of (spacer, rung, and token-bearing) nodes of color gray, black, and orange (C_5 , C_6 , and C_7), there exist exactly two orange (C_7) (token-bearing) nodes not containing tokens, each at distance four from a black (C_6) (rung) node, such that the black (C_6) (rung) nodes are at a distance six from each other. No other black (C_6) (rung) nodes at distance four from an orange (C_7) (token-bearing) node not containing a token exist in the component:

$$\begin{aligned} \forall x \in V(H) : & \left[(C_6(x) \vee C_7(x)) \rightarrow \exists y_1, y_2, u_1, u_2 \in V(H) : \left(y_1 \neq y_2 \wedge \right. \right. \\ & \text{Con}_{5,6,7}(x, y_1) \wedge \text{Con}_{5,6,7}(x, y_2) \wedge \text{Con}_{5,6,7}(x, u_1) \wedge \text{Con}_{5,6,7}(x, u_2) \\ & \wedge u_1 \neq u_2 \wedge \text{CTFC6}(y_1, u_1) \wedge \text{CTFC6}(y_2, u_2) \wedge \text{Dist}_6(u_1, u_2) \\ & \wedge \forall v \in V(H) : \left[(\text{Con}_{5,6,7}(x, v) \wedge C_6(v) \wedge \exists w \in V(H) : \text{CTFC6}(w, v)) \right. \\ & \left. \left. \rightarrow (v = u_1 \vee v = u_2) \right] \right] \end{aligned}$$

(Here, y_1 and y_2 are the orange (token-bearing nodes) and u_1 and u_2 are the black (rung) nodes at distance four from y_1 and y_2 , respectively. The variable x represents any black or orange node and y_1, y_2, u_1, u_2 must exist in the same component as x . Intuitively, B5 ensures that within each connected component of nodes colored gray (C_5), black (C_6), or orange (C_7), there are exactly two black (C_6) nodes that have token-free orange (C_7) nodes at distance four from them, with these two rung nodes separated by distance six. The formula guarantees that no other black (C_6) rung node in the component has a token-free orange (C_7) node at distance four from it.),

and set $\phi(X) = \text{B1}(X) \wedge \text{B2}(X) \wedge \text{B3}(X) \wedge \text{B4}(X) \wedge \text{B5}(X)$.