A Complete Expressiveness Hierarchy for Subgraph GNNs via Subgraph Weisfeiler-Lehman Tests

Bohang Zhang [1] **Guhao Feng** [*2 3] **Yiheng Du** [*4] **Di He** [1] **Liwei Wang** [1 5]

Abstract

Recently, subgraph GNNs have emerged as an important direction for developing expressive graph neural networks (GNNs). While numerous architectures have been proposed, so far there is still a limited understanding of how various design paradigms differ in terms of expressive power, nor is it clear what design principle achieves maximal expressiveness with minimal architectural complexity. To address these fundamental questions, this paper conducts a systematic study of general node-based subgraph GNNs through the lens of Subgraph Weisfeiler-Lehman Tests (SWL). Our central result is to build a *complete hierarchy* of SWL with *strictly* growing expressivity. Concretely, we prove that any node-based subgraph GNN falls into one of the six SWL equivalence classes, among which SSWL achieves the maximal expressive power. We also study how these equivalence classes differ in terms of their practical expressiveness such as encoding graph distance and biconnectivity. In addition, we give a tight expressivity upper bound of all SWL algorithms by establishing a close relation with *localized* versions of WL and Folklore WL (FWL) tests. Overall, our results provide insights into the power of existing subgraph GNNs, guide the design of new architectures, and point out their limitations by revealing an inherent gap with the 2-FWL test. Finally, experiments demonstrate that SSWL-inspired subgraph GNNs can significantly outperform prior architectures on multiple benchmarks despite great simplicity.

[*]Equal contribution [1]National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University [2]School of EECS, Peking University [3]Pazhou Lab [4]Yuanpei College, Peking University [5]Center for Data Science, Peking University. Correspondence to: Bohang Zhang <zhangbohang@pku.edu.cn>, Di He <dihe@pku.edu.cn>, Liwei Wang <wanglw@cis.pku.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Graph neural networks (GNNs), especially equivariant message-passing neural networks (MPNNs), have become the dominant approach for learning on graph-structured data (Gilmer et al., 2017; Hamilton et al., 2017; Kipf & Welling, 2017; Veličković et al., 2018). Despite their great simplicity and scalability, one major drawback of MPNNs lies in the limited expressiveness (Xu et al., 2019; Morris et al., 2019). This motivated a variety of subsequent works to develop provably more expressive architectures, among which subgraph GNNs have emerged as a new trend (Cotta et al., 2021; You et al., 2021; Zhang & Li, 2021; Bevilacqua et al., 2022; Zhao et al., 2022a; Papp & Wattenhofer, 2022; Frasca et al., 2022; Qian et al., 2022; Huang et al., 2022).

Broadly speaking, a general (node-based) subgraph GNN first transforms an input graph G into a collection of subgraphs, each of which is associated with a unique node in G. It then computes a feature representation for each node of each subgraph through a series of equivariant message-passing layers. Finally, it outputs a representation of graph G by pooling all these subgraph node features. Subgraph GNNs have received great attention partly due to their elegant structure, enhanced expressiveness, message-passing-based inductive bias, and superior empirical performance (Frasca et al., 2022; Zhao et al., 2022a).

One central question in subgraph GNNs lies in how to design simple yet expressive equivariant layers. Starting from the most basic design where each node only interacts with its local neighbors in the own subgraph (Cotta et al., 2021; Qian et al., 2022), recent works have developed a rich family of (cross-graph) aggregation operations (Bevilacqua et al., 2022; Zhao et al., 2022a; Frasca et al., 2022). In particular, Frasca et al. (2022) gave a unified characterization of the design space of subgraph GNNs based on 2-IGN (Maron et al., 2019b;a), which contains *dozens* of atomic aggregations. However, it is generally unclear whether the added aggregations can *theoretically* improve a model's expressiveness as it becomes increasingly complex. So far, a systematic investigation and comparison of various possible aggregation schemes in terms of expressiveness is still lacking. More fundamentally, for both theory and practice, *is there a canonical design principle of subgraph GNNs that achieves the maximal expressiveness with the least model complexity?*

A complete hierarchy of subgraph GNNs. In this paper, we comprehensively study the aforementioned questions through the lens of Subgraph Weisfeiler-Lehman Tests (SWL), a class of color refinement algorithms abstracted from subgraph GNNs in distinguishing non-isomorphic graphs. Each SWL consists of three ingredients: (a) graph generation policy, (b) message-passing aggregation scheme, and (c) final pooling scheme. Among commonly used graph generation policies, we mainly focus on the canonical *node marking* SWL as it theoretically achieves the best expressive power despite simplicity (Proposition 4.2). Our central result is to build a complete hierarchy for all node marking SWL with various aggregation schemes and pooling schemes. Concretely, we prove that any node-based subgraph GNN falls into one of the six SWL equivalence classes and establish *strict expressivity inclusion relationships* between different classes (see Corollary 4.7, Theorem 7.1, and Figure 1). In particular, our result highlights that, by including symmetrically two basic local aggregations, the corresponding SWL (called SSWL) has theoretically achieved the maximal expressive power. Our result thus provides a clear picture of the power and limitation of existing architectures, settling a series of open problems raised in Bevilacqua et al. (2022); Frasca et al. (2022); Qian et al. (2022); Zhao et al. (2022a) (see Section 8 for discussions).

Related to practical expressiveness. We provide concrete evidence that subgraph GNNs with better theoretical expressivity are also stronger in terms of their ability to compute fundamental graph properties. Inspired by the recent work of Zhang et al. (2023), we prove that the PSWL (defined in Corollary 4.7) is *strictly more powerful* than a variant of the Generalized Distance WL proposed in their paper, which incorporates both the *shortest path distance* and the *hitting time distance* (Definition A.1). Our result *unifies* and *extends* the findings in Zhang et al. (2023) and implies that all SWL algorithms stronger than PSWL are able to encode both *distance* and *biconnectivity* properties. In contrast, we give counterexamples to show that *neither* of these basic graph properties can be fully encoded in vanilla SWL.

Localized (Folklore) WL tests. Similar to the classic WL and Folklore WL tests (Weisfeiler & Leman, 1968; Cai et al., 1992), node marking SWL corresponds to a natural class of computation models for graph canonization (Immerman & Lander, 1990). All SWL algorithms have $O(n^2)$ memory complexity and $O(nm)$ computational complexity (per iteration) for a graph of n vertices and m edges. Owing to the improved computational efficiency over classic 2-FWL/3-WL (i.e. $O(n^3)$), a better understanding of what can/cannot be achieved under this complexity class is arguably an important research question. We answer this question by first establishing a close relation between SWL and *localized* versions of 2-WL (Morris et al., 2020) and 2-FWL tests, both of which have the same complexity as SWL. We then

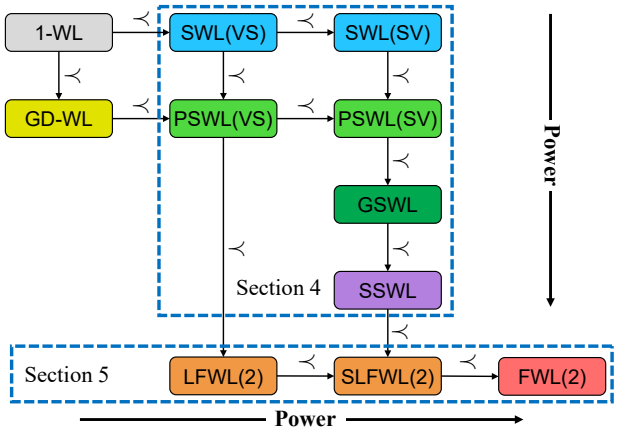


Figure 1. Expressiveness hierarchy of different WL algorithms.

derive a number of key results: (i) The strongest SSWL is *as powerful as* localized 2-WL. This builds a surprising link between the works of Frasca et al. (2022) and Morris et al. (2020). (ii) Despite the same complexity, there is an *inherent gap* between localized 2-WL and localized 2-FWL. (iii) There is an *inherent gap* between localized 2-FWL and classic 2-FWL. Consequently, our results settle a fundamental open problem raised in Frasca et al. (2022) about whether subgraph GNNs can match the power of 2-FWL, and further implies that *subgraph GNNs even do not reach the maximal expressiveness in the model class of $O(nm)$ complexity*. This reveals an intrinsic limitation of the subgraph GNN model class and points out a new direction for improvement.

Technical Contributions. Actually, it is quite challenging to find a principled class of hard graphs that can reveal the expressivity gap of different SWL/FWL-type algorithms. As a main technical contribution, we develop a novel analyzing framework inspired by Cai et al. (1992) based on *pebbling games*, where we considerably extend the game originally designed for FWL to all types of SWL and localized 2-WL/2-FWL algorithms. The game viewpoint offers deep insights into the power of different algorithms, through which we can skillfully construct a collection of nontrivial counterexample graphs to prove all strict separation results in this paper. We believe the proposed games and counterexamples may be of independent value in future work.

Practical Contributions. Our theoretical insights can also guide in designing simple, efficient, yet powerful subgraph GNN architectures. In particular, the proposed SSWL corresponds to an elegant design principle with only 3 atomic equivariant aggregation operations, yet the resulting model is strictly more powerful than all prior node-based subgraph GNNs. Empirically, we verify SSWL-based subgraph GNNs on several benchmarks such as substructure counting and molecular property prediction, showing that they can significantly outperform prior architectures despite fewer model parameters and great simplicity.

2. Formalizing Subgraph GNNs

Notations. We use $\{\ \}$ to denote sets and use $\{\!\{\ \}\!\}$ to denote multisets. The cardinality of (multi)set $\mathcal{S}$ is denoted as $|\mathcal{S}|$. In this paper, we consider finite, undirected, simple, *connected* graphs, and we use $G = (\mathcal{V}_G, \mathcal{E}_G)$ to denote such a graph with vertex set $\mathcal{V}_G$ and edge set $\mathcal{E}_G$. Each edge in $\mathcal{E}_G$ is expressed as a set $\{u, v\}$ containing two distinct vertices in $\mathcal{V}_G$. Given a vertex u, denote its *neighbors* as $\mathcal{N}_G(u) := \{v \in \mathcal{V}_G : \{u, v\} \in \mathcal{E}_G\}$. Similarly, the *$k$-hop neighbors* of u is denoted as $\mathcal{N}_G^k(u) := \{v \in \mathcal{V}_G : \mathrm{dis}_G(u, v) \leq k\}$, where $\mathrm{dis}_G(u, v)$ is the shortest path distance between u and v. In particular, $\mathcal{N}_G^1(u) = \mathcal{N}_G(u) \cup \{u\}$.

A general subgraph GNN processes an input graph G following three steps: (i) generating subgraphs, (ii) equivariant message-passing, and (iii) final pooling. Below, we separately describe each of these components.

Node-based graph generation policies. The first step is to generate a collection of subgraphs of G based on a predefined graph generation policy π and initialize node features in each subgraph. For *node-based* subgraph GNNs, there are a total of $|\mathcal{V}_G|$ subgraphs, and each subgraph is uniquely associated with a specific node $u \in \mathcal{V}_G$, so that π can be expressed as a mapping of the form $\pi(G) = \{\!\{(G^u, \tilde{h}_G^u) : u \in \mathcal{V}_G\}\!\}$. Here, all subgraphs $G^u = (\mathcal{V}_G, \mathcal{E}_G^u)$ share the vertex set $\mathcal{V}_G$ but may differ in the edge set $\mathcal{E}_G^u$. The mapping $\tilde{h}_G^u : \mathcal{V}_G \to \mathbb{R}^d$ defines the initial node features, i.e., $\tilde{h}_G^u(v)$ is the initial feature of vertex v in subgraph G^u.

Various graph generation policies have been proposed in prior works, which differ in the choice of $\mathcal{E}_G^u$ and $\tilde{h}_G^u$. For example, common choices of $\mathcal{E}_G^u$ are: (i) using the original graph ($\mathcal{E}_G^u = \mathcal{E}_G$), (ii) node deletion ($\mathcal{E}_G^u = \mathcal{E}_G \backslash \{\{u, v\} : v \in \mathcal{N}_G(u)\}$ by deleting all edges associated to node u, and (iii) k-hop ego network ($\mathcal{E}_G^u = \{\{v, w\} \in \mathcal{E}_G : v, w \in \mathcal{N}_G^k(u)\}$). To initialize node features, there are also three popular choices: (i) constant node features, where $\tilde{h}_G^u(v)$ is the same for all $u, v \in \mathcal{V}_G$; (ii) node marking, where $\tilde{h}_G^u(v)$ depends only on whether $u = v$ or not; (iii) distance encoding, where $\tilde{h}_G^u(v)$ depends on the shortest path distance between u and v, i.e. $\mathrm{dis}_G(u, v)$.

In this paper, we mainly consider the canonical *node marking* policy on the original graph due to its simplicity. Importantly, we will show in Section 4.1 that it already achieves the maximal expressiveness among all the above policies.

Equivariant message-passing. The main backbone of subgraph GNNs consists of L stacked equivariant message-passing layers. For each network layer $l \in [L]$, the feature of each node v in each subgraph G^u is computed, which can be denoted as $h_G^{(l)}(u, v)$. At the beginning, $h_G^{(0)}(u, v) = \tilde{h}_G^u(v)$. Following Frasca et al. (2022), we study arguably the most general design space that incorporates a broad class of possible message-passing aggregation operations.

Definition 2.1. A general subgraph GNN layer has the form

$$h_G^{(l+1)}(u,v) = \sigma^{(l+1)}(\mathsf{op}_1(u,v,G,h_G^{(l)}), \cdots, \mathsf{op}_r(u,v,G,h_G^{(l)})),$$

where $\sigma^{(l+1)}$ is an arbitrary (parameterized) continuous function, and each atomic operation $\mathsf{op}_i(u, v, G, h)$ can take any of the following expressions:

- Single-point: $h(u, v)$, $h(v, u)$, $h(u, u)$, or $h(v, v)$;
- Global: $\sum_{w \in \mathcal{V}_G} h(u, w)$ or $\sum_{w \in \mathcal{V}_G} h(w, v)$;
- Local: $\sum_{w \in \mathcal{N}_{G^u}(v)} h(u, w)$ or $\sum_{w \in \mathcal{N}_{G^v}(u)} h(w, v)$.

We assume that $h(u, v)$ is always present in some op_i.

It is easy to see that any GNN layer defined above is permutation *equivariant*. Among them, two most basic atomic operations are $h(u, v)$ and $\sum_{w \in \mathcal{N}_{G^u}(v)} h(u, w)$, which are applied in all prior subgraph GNNs. Without using further operations, the *vanilla subgraph GNN* layer has the form

$$h_G^{(l+1)}(u, v) = \sigma^{(l+1)}\left(h_G^{(l)}(u, v), \sum_{w \in \mathcal{N}_{G^u}(v)} h_G^{(l)}(u, w)\right).$$

Besides, several works have explored other aggregation operations, and we list a few representative examples below[1].

Example 2.2. (i) ESAN (Bevilacqua et al., 2022) additionally uses global aggregation $\sum_{w \in \mathcal{V}_G} h(w, v)$. (ii) GNN-AK (Zhao et al., 2022a) additionally uses single-point operation $h(v, v)$. It also uses global aggregation $\sum_{w \in \mathcal{V}_G} h(u, w)$ when $u = v$. (iii) SUN (Frasca et al., 2022) additionally uses $h(u, u)$, $h(v, v)$, and both types of global aggregations.

Final pooling layer. The last step is to output a graph representation $f(G)$ based on all the collected features $\{\!\{h_G^{(L)}(u, v) : u, v \in \mathcal{V}_G\}\!\}$. There are two different ways to implement this, which differ in the order of pooling along the two dimensions u, v. The first approach, called *vertex-subgraph pooling*, first pools all node features in each subgraph G^u to obtain the subgraph representation, i.e., $f^{\mathsf{S}}(G, u) := \sigma^{\mathsf{S}}\left(\sum_{v \in \mathcal{V}_G} h_G^{(L)}(u, v)\right)$, and then pools all subgraph representations to obtain the final output $f(G) := \sigma^{\mathsf{G}}(\sum_{u \in \mathcal{V}_G} f^{\mathsf{S}}(G, u))$. Here, σ^{S} and σ^{G} can be any parameterized function. Most prior works follow this paradigm. In contrast, the second approach, called

[1]We note that there are still other possible equivariant operations that are not included in Definition 2.1, such as diagonal aggregations (e.g., $\sum_{w \in \mathcal{V}_G} h(w, w)$) and composite aggregations (e.g., $\sum_{w \in \mathcal{V}_G} \sum_{x \in \mathcal{N}_G(v)} h(w, x)$ used in ESAN). In particular, Frasca et al. (2022) recently proposed a powerful subgraph GNN framework called RelGN(2), which contains a total of 39 atomic operations for node-marking policy (the number can be even larger for other policies). However, we prove that incorporating these operations does not bring extra expressiveness beyond the current framework (see Appendix F). Here, we select the 8 basic operations in Definition 2.1 mainly due to their fundamental nature, simplicity, and completeness.

subgraph-vertex pooling, first generates node representations $f^{\mathsf{V}}(G,v) := \sigma^{\mathsf{V}}(\sum_{u\in\mathcal{V}_G} h_G^{(L)}(u,v))$ for each $v \in \mathcal{V}_G$, and then pools all these node representations to obtain the graph representation, i.e., $f(G) := \sigma^{\mathsf{G}}(\sum_{v\in\mathcal{V}_G} f^{\mathsf{V}}(G,v))$. This approach is adopted in Qian et al. (2022).

3. Subgraph Weisfeiler-Lehman Test

To formally study the expressive power of subgraph GNNs, in this section we introduce the Subgraph WL Test (SWL), a class of color refinement algorithms for graph isomorphism test. Let $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ be two graphs. As with subgraph GNNs, SWL first generates for each graph a collection of subgraphs and initializes color mappings based on a graph generation policy π. We denote the results as $\{\!\{(G^u, \tilde{\chi}_G^u) : u \in \mathcal{V}_G\}\!\}$ and $\{\!\{(H^u, \tilde{\chi}_H^u) : u \in \mathcal{V}_H\}\!\}$, where $\tilde{\chi}$ is the color mapping that can be constant, node marking or distance encoding (according to Section 2).

Given graph G, let $\chi_G^{(0)}(u,v) := \tilde{\chi}_G^u(v)$ for $u, v \in \mathcal{V}_G$. SWL then refines the color of each (u,v) pair using various types of aggregation operations defined as follows:

Definition 3.1. A general SWL iteration has the form

$$\chi_G^{(t+1)}(u,v) = \mathsf{hash}(\mathsf{agg}_1(u,v,G,\chi_G^{(t)}),\cdots,\mathsf{agg}_r(u,v,G,\chi_G^{(t)})),$$

where hash is a perfect hash function and each $\mathsf{agg}_i(u,v,G,\chi)$ can take any of the following expressions:

- Single-point: $\chi(u,v)$, $\chi(v,u)$, $\chi(u,u)$, or $\chi(v,v)$;
- Global: $\{\!\{\chi(u,w) : w \in \mathcal{V}_G\}\!\}$ or $\{\!\{\chi(w,v) : w \in \mathcal{V}_G\}\!\}$.
- Local: $\{\!\{\chi(u,w) : w \in \mathcal{N}_{G^u}(v)\}\!\}$ or $\{\!\{\chi(w,v) : w \in \mathcal{N}_{G^v}(u)\}\!\}$.

We use symbols $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, and $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$ to denote each of the 8 basic operations, respectively. We assume $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ is always present in some agg_i. The set $\mathcal{A} := \{\mathsf{agg}_i : i \in [r]\}$ fully determines the SWL iteration and is called the *aggregation scheme*.

For each iteration t, the color mapping $\chi_G^{(t)}$ induces an equivalence relation and thus a *partition* $\mathcal{P}_G^{(t)}$ over the set $\mathcal{V}_G \times \mathcal{V}_G$. Since $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ is present in $\mathcal{A}$, $\mathcal{P}_G^{(t)}$ must get *refined* as t grows. Therefore, with a sufficiently large number of iterations $t \le |\mathcal{V}_G|^2$, the color mapping becomes stable (i.e., inducing a *stable partition*). Without abuse of notation, we denote the stable color mapping by χ_G.

Finally, the representation of graph G, denoted as $c(G)$, is computed by hashing all colors $\chi_G(u,v)$ for $u, v \in \mathcal{V}_G$. Parallel to the previous section, there are two different *pooling paradigms* to implement this:

- Vertex-subgraph pooling (abbreviated as VS): $c(G) = \mathsf{hash}\left(\{\!\{\mathsf{hash}(\{\!\{\chi_G(u,v) : v \in \mathcal{V}_G\}\!\}) : u \in \mathcal{V}_G\}\!\}\right)$;
- Subgraph-vertex pooling (abbreviated as SV): $c(G) = \mathsf{hash}\left(\{\!\{\mathsf{hash}(\{\!\{\chi_G(u,v) : u \in \mathcal{V}_G\}\!\}) : v \in \mathcal{V}_G\}\!\}\right)$.

We say SWL can distinguish a pair of graphs G and H if $c(G) \neq c(H)$. Similarly, given a subgraph GNN f, we say f distinguishes graphs G and H if $f(G) \neq f(H)$. The following proposition establishes the connection between SWL and subgraph GNNs in terms of expressivity in distinguishing non-isomorphic graphs.

Proposition 3.2. *The expressive power of any subgraph GNN defined in Section 2 is bounded by a corresponding SWL by matching the graph generation policy π, the aggregation scheme (between Definitions 2.1 and 3.1), and the pooling paradigm. Moreover, when considering bounded-size graphs, for any SWL algorithm, there exists a matching subgraph GNN with the same expressive power.*

Qian et al. (2022) first proved the above result for *vanilla* subgraph GNNs without cross-graph aggregations. Here, we consider general aggregation schemes and give a unified proof of Proposition 3.2 in Appendix D. Based on this result, we can focus on studying the expressive power of SWL in subsequent analysis.

4. Expressiveness and Hierarchy of SWL

In this section, we systematically study how different design paradigms impact the expressiveness of SWL algorithms. To begin with, we need the following set of terminologies:

Definition 4.1. Let A_1 and A_2 be two color refinement algorithms, and denote $c_i(G)$, $i \in \{1,2\}$ as the graph representation computed by A_i for graph G. We say:

- A_1 is *more powerful* than A_2, denoted as $\mathsf{A}_2 \preceq \mathsf{A}_1$, if for any pair of graphs G and H, $c_1(G) = c_1(H)$ implies $c_2(G) = c_2(H)$.
- A_1 is *as powerful as* A_2, denoted as $\mathsf{A}_1 \simeq \mathsf{A}_2$, if both $\mathsf{A}_1 \preceq \mathsf{A}_2$ and $\mathsf{A}_2 \preceq \mathsf{A}_1$ hold.
- A_1 is *strictly more powerful* than A_2, denoted as $\mathsf{A}_2 \prec \mathsf{A}_1$, if $\mathsf{A}_2 \preceq \mathsf{A}_1$ and $\mathsf{A}_2 \not\simeq \mathsf{A}_1$, i.e., there exist graphs G, H such that $c_1(G) \neq c_1(H)$ and $c_2(G) = c_2(H)$.
- A_1 and A_2 are *incomparable*, denoted as $\mathsf{A}_1 \nsim \mathsf{A}_2$, if neither $\mathsf{A}_1 \preceq \mathsf{A}_2$ nor $\mathsf{A}_2 \preceq \mathsf{A}_1$ holds.

4.1. The canonical form: node marking SWL test

The presence of many different graph generation policies complicates our subsequent analysis. Interestingly, however, we show the simple *node marking* policy (on the original graph) already achieves the maximal power among all policies considered in Section 2 under mild assumptions.

Proposition 4.2. *Consider any SWL algorithm A that contains the two basic aggregations $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ and $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ in Definition 3.1. Denote $\hat{\mathsf{A}}$ as the corresponding algorithm obtained from A by replacing the graph generation policy π to node marking (on the original graph). Then, $\mathsf{A} \preceq \hat{\mathsf{A}}$.*

We give a proof in Appendix E.2, which is based on the following finding: when the special node mark is propagated by SWL with local aggregation, the color of each node pair (u, v) can encode its distance $\mathrm{dis}_G(u, v)$ (Lemma E.4), and the structure of k-hop ego network is also encoded.

Note that for the node marking policy, all subgraphs are just the original graph ($G^u = G$), which simplifies our analysis. We hence focus on the simple yet expressive node marking policy in subsequent sections. The following notations will be frequently used:

Definition 4.3. Denote $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ as the node marking SWL test with aggregation scheme $\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}\}$ and pooling paradigm Pool, where $\mathsf{Pool} \in \{\mathsf{VS}, \mathsf{SV}\}$, and

$$\mathcal{A} \subset \{\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}.$$

Here, we assume that $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ is always present in SWL.

4.2. Hierarchy of different algorithms

As shown in Definition 4.3, there are a large number of possible combinations of aggregation/pooling designs. In this subsection, we aim to build a complete hierarchy of SWL algorithms by establishing expressivity inclusion relations between different design paradigms. All proofs in this section are deferred to Appendix E.

We first consider the expressive power of different aggregation schemes. We have the following main theorem:

Theorem 4.4. *Under the notation of Definition 4.3, for any $\mathcal{A}$ and Pool, the following hold:*

- $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{Pool})$ *and* $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}\}, \mathsf{Pool})$*;*
- $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}\}, \mathsf{Pool})$ *and* $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}\}, \mathsf{Pool})$*;*
- $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool})$.

Theorem 4.4 shows that local aggregation is more powerful than (and can *express*) the corresponding global aggregation, while global aggregation is more powerful than (and can *express*) the corresponding single-point aggregation. In addition, the "transpose" aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$ is quite powerful: when combining a local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, it can express the other local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$.

We next turn to the pooling paradigm. We first show that there is a symmetry (duality) between u, v and the two types of pooling paradigms VS, SV.

Proposition 4.5. *Let $\mathcal{A}$ be any aggregation scheme defined in Definition 4.3. Denote $\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}$ as the aggregation scheme obtained from $\mathcal{A}$ by exchanging the element $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$ with $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, exchanging $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$ with $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, and exchanging $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ with $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$. Then, $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \simeq \mathsf{A}(\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}, \mathsf{SV})$.*

Based on the symmetry, one can easily extend Theorem 4.4 to a variant that gives relations for $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, and $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$. Moreover, we have the following main theorem:

Theorem 4.6. *Let $\mathcal{A}$ be defined in Definition 4.3 with $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}} \in \mathcal{A}$. Then, the following hold:*

- $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{SV})$*;*
- *If* $\{\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\} \cap \mathcal{A} \neq \emptyset$*, then* $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \simeq \mathsf{A}(\mathcal{A}, \mathsf{SV})$.

Theorem 4.6 indicates that the subgraph-vertex pooling is always more powerful than the vertex-subgraph pooling, especially when the aggregation scheme is weak (e.g, the vanilla SWL). On the other hand, they become equally expressive for SWL with strong aggregation schemes.

Combined with the above three results, we have built a *complete hierarchy* for the expressive power of all node marking SWL algorithms in Definition 4.3. In particular, we show any SWL must fall into the following 6 types:

Corollary 4.7. *Let $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ be any SWL defined in Definition 4.3 with at least one local aggregation, i.e. $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\} \cap \mathcal{A} \neq \emptyset$. Then, $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ must be as expressive as one of the 6 SWL algorithms defined below:*

- (Vanilla SWL) $\mathsf{SWL}(\mathsf{VS}) := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{VS})$, $\mathsf{SWL}(\mathsf{SV}) := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{SV})$;
- (SWL with additional single-point aggregation) $\mathsf{PSWL}(\mathsf{VS}) := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{VS})$, $\mathsf{PSWL}(\mathsf{SV}) := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{SV})$;
- (SWL with additional global aggregation) $\mathsf{GSWL} := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{VS})$;
- (Symmetrized SWL) $\mathsf{SSWL} := \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{VS})$.

Moreover, we have

$$\mathsf{SWL}(\mathsf{VS}) \preceq \mathsf{SWL}(\mathsf{SV}) \text{ and } \mathsf{PSWL}(\mathsf{VS}) \preceq \mathsf{PSWL}(\mathsf{SV}),$$
$$\mathsf{SWL}(\mathsf{VS}) \preceq \mathsf{PSWL}(\mathsf{VS}) \text{ and } \mathsf{SWL}(\mathsf{SV}) \preceq \mathsf{PSWL}(\mathsf{SV}),$$
$$\mathsf{PSWL}(\mathsf{SV}) \preceq \mathsf{GSWL} \preceq \mathsf{SSWL}.$$

Corollary 4.7 is significant in that it drastically reduces the problem of studying a large number of different SWL variants to the study of only 6 standard paradigms. Moreover, it implies that the simple SSWL already achieves the maximal expressive power among all SWL variants. A detailed discussion on how these standard paradigms relate to previously proposed subgraph GNNs will be made in Section 8.

Yet, there are still two fundamental problems that are not answered in Corollary 4.7. *First*, it remains unclear whether some SWL algorithm is *strictly* more powerful than another. This question is particularly important for a better understanding of how global, local, and single-point aggregations vary in their expressive power brought to SWL.

Second, a deep understanding of the *limitation* of SWL algorithms is still open. While Frasca et al. (2022); Qian

et al. (2022) recently discovered that the expressiveness of subgraph GNNs can be upper bounded by the standard 2-FWL (3-WL) test, it remains a mystery whether there is an inherent gap between 2-FWL and SWL (in particular, the strongest SSWL). Note that the per-iteration complexity of SWL is $O(nm)$ for a graph of n vertices and m edges, which is remarkably lower than 2-FWL ($O(n^3)$ complexity), so it is reasonable to expect that 2-FWL is strictly more powerful. If this is the case, one may further ask: *does SWL achieve the maximal power among all color refinement algorithms with complexity $O(nm)$*? We aim to fully address both the above fundamental questions in subsequent sections.

5. Localized Folklore Weisfeiler-Lehman Test

In this section, we propose two novel types of WL algorithms based on the standard 2-dimensional Folklore Weisfeiler-Lehman test (2-FWL) (Weisfeiler & Leman, 1968; Cai et al., 1992), which turns out to be closely related to SWL. Recall that 2-FWL maintains a color for each vertex pair $(u, v) \in \mathcal{V}_G \times \mathcal{V}_G$. Initially, the color $\chi_G^{(0)}(u, v)$ depends on the isomorphism type of the subgraph induced by (u, v), namely, depending on whether $u = v$, $\{u, v\} \in \mathcal{E}_G$, or $\{u, v\} \notin \mathcal{E}_G$. In each iteration t, the color is refined by the following update formula:

$$\chi_G^{(t+1)}(u, v) = \mathsf{hash}(\chi_G^{(t)}(u, v), \mathsf{walk}(u, v, \mathcal{V}_G, \chi_G^{(t)})), \tag{1}$$

where we define

$$\mathsf{walk}(u, v, \mathcal{V}, \chi) := \{\!\{(\chi(u, w), \chi(w, v)) : w \in \mathcal{V}\}\!\}. \tag{2}$$

The color mapping $\chi_G^{(t)}$ stabilizes after a sufficiently large number of iterations $t \leq |\mathcal{V}_G|^2$. Denote the stable color mapping as χ_G. 2-FWL finally outputs the graph representation $c(G) := \mathsf{hash}(\{\!\{\chi_G(u, v) : u, v \in \mathcal{V}_G\}\!\})$.

One can see that each 2-FWL iteration has a complexity of $O(n^3)$ for a graph of n vertices and m edges, due to the need to enumerate all $w \in \mathcal{V}_G$ for each pair (u, v). For sparse graphs where $m = o(n^2)$, 2-FWL is inefficient and does not well-exploit the sparse nature of the graph. This inspires us to consider variants of 2-FWL that enumerate only the *local neighbors*, such as $w \in \mathcal{N}_G^1(v)$, by which the rich adjacency information is naturally incorporated in the *update formula* (besides in the initial colors by the isomorphism type). We note that such an idea was previously explored in Morris et al. (2020) (see Section 8 for further discussions). Importantly, the simple change substantially reduces the computational cost to $O(nm)$, which is the same as SWL. To this end, we define two novel FWL-type algorithms:

Definition 5.1. Define $\mathsf{LFWL}(2)$ as the localized version of 2-FWL, which replaces $\mathcal{V}_G$ in (1) by $\mathcal{N}_G^1(v)$. Define $\mathsf{SLFWL}(2)$ as the symmetrized version of $\mathsf{LFWL}(2)$, which replaces $\mathcal{V}_G$ in (1) by $\mathcal{N}_G^1(u) \cup \mathcal{N}_G^1(v)$. Finally, denote $\mathsf{FWL}(2)$ as the standard 2-FWL for consistency.

Note that $\mathsf{LFWL}(2)$ only exploits the local information of the vertex v, while $\mathsf{SLFWL}(2)$ uses all the local information of a vertex pair (u, v) while still maintaining the $O(nm)$ cost. Therefore, one may expect that the latter is more powerful. Indeed, we have the following central result:

Theorem 5.2. *The following relations hold:*

- $\mathsf{LFWL}(2) \preceq \mathsf{SLFWL}(2) \preceq \mathsf{FWL}(2)$*;*
- $\mathsf{PSWL}(\mathsf{VS}) \preceq \mathsf{LFWL}(2)$ *and* $\mathsf{SSWL} \preceq \mathsf{SLFWL}(2)$.

The proof is given in Appendix G. We now make several discussions regarding the significance of Theorem 5.2. *First*, $\mathsf{FWL}(2)$ is more powerful than its localized variants, confirming that there is indeed a trade-off between complexity and expressiveness. *Second*, Theorem 5.2 reveals a close relationship between SWL and these localized 2-WL/2-FWL variants. In particular, $\mathsf{SLFWL}(2)$ is more powerful than all SWL algorithms *despite the same computational cost*. Therefore, we obtain a *tight* upper bound on the expressive power of subgraph GNNs with *matching* complexity, which remarkably improves the previous 2-FWL upper bound (Frasca et al., 2022; Qian et al., 2022).

However, again, it is not known whether these localized 2-FWL variants are *strictly more powerful* than SWL, nor do we know whether there is an intrinsic gap between 2-FWL and its localized variants. To thoroughly answer all of these questions, we need a new tool: the *pebbling game*.

6. Pebbling Game

In this section, we develop a novel and unified analyzing framework for various SWL/FWL algorithms based on Ehrenfeucht-Fraïssé games (Ehrenfeucht, 1961; Fraïsaé, 1954). The seminal paper of Cai et al. (1992) has used such games to prove the existence of counterexample graphs which k-FWL could not distinguish. Here, we vastly extend their result and show how pebbling games can be used to analyze *all types* of SWL and localized FWL algorithms. All proofs in this section are deferred to Appendix H.

First consider any SWL algorithm $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$. The pebbling game is played on two graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$. Each graph is equipped with two different pebbles u and v, both of which lie outside the graph initially. There are two players, the *Spoiler* and the *Duplicator*. To describe the game, we first introduce a basic game operation dubbed "vertex selection".

Definition 6.1 (Vertex Selection)**.** Let $\mathcal{S}_G \subset \mathcal{V}_G$ and $\mathcal{S}_H \subset \mathcal{V}_H$ be given sets. Spoiler first freely chooses a non-empty subset $\mathcal{S}^{\mathsf{S}}$ from either $\mathcal{S}_G$ or $\mathcal{S}_H$, and Duplicator should respond with a subset $\mathcal{S}^{\mathsf{D}}$ from the other set, satisfying $|\mathcal{S}^{\mathsf{S}}| = |\mathcal{S}^{\mathsf{D}}|$. Duplicator loses the game if she has no feasible choice. Then, Spoiler can select any vertex $x^{\mathsf{S}} \in \mathcal{S}^{\mathsf{D}}$, and Duplicator responds by selecting any vertex $x^{\mathsf{D}} \in \mathcal{S}^{\mathsf{S}}$.

Initialization. If $\mathsf{Pool} = \mathsf{VS}$, the two players first select vertices x^{S} and x^{D} following the vertex selection procedure with $\mathcal{S}_G = \mathcal{V}_G$ and $\mathcal{S}_H = \mathcal{V}_H$. Spoiler places pebble u on the selected vertex x^{S} and Duplicator places the other pebble u on vertex x^{D}. Next, Spoiler and Duplicator perform the vertex selection step again with $\mathcal{S}_G = \mathcal{V}_G$ and $\mathcal{S}_H = \mathcal{V}_H$ and place pebbles v similarly. If $\mathsf{Pool} = \mathsf{SV}$, the above procedure is analogous except that Spoiler/Duplicator places pebble v in the first step and pebble u in the second step.

Main loop. The game then cyclically executes the following process. Depending on the SWL aggregation scheme $\mathcal{A}$, Spoiler can freely choose one of the following ways to play:

- Local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}} \in \mathcal{A}$. Spoiler and Duplicator perform the vertex selection step with $\mathcal{S}_G = \mathcal{N}_G(v)$ and $\mathcal{S}_H = \mathcal{N}_H(v)$, where $\mathcal{N}_G(v)/\mathcal{N}_H(v)$ represents the set of vertices in graph G/H adjacent to the vertex placed by pebble v. Spoiler moves pebble v to the selected vertex x^{S}, and Duplicator moves the other pebble v to vertex x^{D}.
- Global aggregation $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}} \in \mathcal{A}$. Spoiler and Duplicator perform the vertex selection step with $\mathcal{S}_G = \mathcal{V}_G$ and $\mathcal{S}_H = \mathcal{V}_H$. Spoiler moves pebble v to the selected vertex x^{S}, and Duplicator moves the other pebble v to vertex x^{D}.
- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}} \in \mathcal{A}$. Both players move pebble v to the position of pebble u.
- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}} \in \mathcal{A}$. Both players swap the position of pebbles u and v.

The cases of $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$ are similar (symmetric) to $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$, so we omit them for clarity.

Spoiler wins the game if, after a certain round, the subgraph of G induced by vertices placed by pebbles u, v does not have the same isomorphism type as that of H. Duplicator wins the game if Spoiler cannot win after any number of rounds. Roughly speaking, Spoiler tries to find differences between graphs G and H using pebbles u and v, while Duplicator strives to make these pebbles look the same in the two graphs. Our main result is stated as follows:

Theorem 6.2. *Let* $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ *be any SWL algorithm defined in Definition 4.3, satisfying* $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\} \cap \mathcal{A} \neq \emptyset$. *Then,* $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ *can distinguish a pair of graphs G and H if and only if Spoiler can win the corresponding pebbling game on graphs G and H.*

We next turn to FWL-type algorithms. The games are mostly similar to SWL but with a few subtle differences. There are also two pebbles u, v for each graph. Here, the two players first places pebbles u, v using just *one* vertex selection step: Spoiler first chooses a non-empty subsets $\mathcal{S}^{\mathsf{S}}$ from either $\mathcal{V}_G \times \mathcal{V}_G$ or $\mathcal{V}_H \times \mathcal{V}_H$, and Duplicator should respond with a subset $\mathcal{S}^{\mathsf{D}}$ from the other set, satisfying $|\mathcal{S}^{\mathsf{S}}| = |\mathcal{S}^{\mathsf{D}}|$. Then, Spoiler selects any vertex pair $(x^{\mathsf{S}}_{\mathsf{u}}, x^{\mathsf{S}}_{\mathsf{v}}) \in \mathcal{S}^{\mathsf{D}}$, and Duplicator responds by selecting $(x^{\mathsf{D}}_{\mathsf{u}}, x^{\mathsf{D}}_{\mathsf{v}}) \in \mathcal{S}^{\mathsf{S}}$. Spoiler places pebbles u and v on $x^{\mathsf{S}}_{\mathsf{u}}$ and $x^{\mathsf{S}}_{\mathsf{v}}$, respectively. Duplicator places the other pebbles u and v on $x^{\mathsf{D}}_{\mathsf{u}}$ and $x^{\mathsf{D}}_{\mathsf{v}}$, respectively.

The game then cyclically executes the following process. First consider $\mathsf{LFWL}(2)$. In each round, the two players perform the vertex selection step with $\mathcal{S}_G = \mathcal{N}^1_G(v)$ and $\mathcal{S}_H = \mathcal{N}^1_H(v)$ and select vertices x^{S} and x^{D}, respectively. Then it comes to the major difference from SWL: Spoiler can choose *whether* to move pebble u *or* pebble v to vertex x^{S}, and Duplicator should move the same pebble in the other graph to x^{D}. For $\mathsf{SLFWL}(2)$, the process is exactly the same as above except that the vertex selection is performed with $\mathcal{S}_G = \mathcal{N}^1_G(u) \cup \mathcal{N}^1_G(v)$ and $\mathcal{S}_H = \mathcal{N}^1_H(u) \cup \mathcal{N}^1_H(v)$. Finally, for the standard $\mathsf{FWL}(2)$, the vertex selection is performed with $\mathcal{S}_G = \mathcal{V}_G$ and $\mathcal{S}_H = \mathcal{V}_H$. We have the following result:

Theorem 6.3. $\mathsf{LFWL}(2)/\mathsf{SLFWL}(2)/\mathsf{FWL}(2)$ *can distinguish a pair of graphs G and H if and only if Spoiler can win the corresponding pebbling game on graphs G and H.*

Theorems 6.2 and 6.3 build an interesting connection between WL algorithms and games. Importantly, the game viewpoint offers us a much clearer picture to sort out various complex aggregation/pooling paradigms and leads to the main result of this paper in the next section.

7. Strict Separation Results

Up to now, all results derived in this paper are of the form "$\mathsf{A}_1 \preceq \mathsf{A}_2$". In this section, we will complete the analysis by proving that all relations $\preceq$ in Corollary 4.7 and Theorem 5.2 are actually the strict relations $\prec$. Formally, we will prove:

Theorem 7.1. *The following hold:*

- $\mathsf{SWL}(\mathsf{VS}) \prec \mathsf{SWL}(\mathsf{SV})$, $\mathsf{PSWL}(\mathsf{VS}) \prec \mathsf{PSWL}(\mathsf{SV})$;
- $\mathsf{SWL}(\mathsf{VS}) \prec \mathsf{PSWL}(\mathsf{VS})$, $\mathsf{SWL}(\mathsf{SV}) \prec \mathsf{PSWL}(\mathsf{SV})$;
- $\mathsf{PSWL}(\mathsf{SV}) \prec \mathsf{GSWL} \prec \mathsf{SSWL}$;
- $\mathsf{PSWL}(\mathsf{VS}) \prec \mathsf{LFWL}(2)$, $\mathsf{SSWL} \prec \mathsf{SLFWL}(2)$;
- $\mathsf{LFWL}(2) \prec \mathsf{SLFWL}(2) \prec \mathsf{FWL}(2)$;
- $\mathsf{SWL}(\mathsf{SV}) \nprec \mathsf{PSWL}(\mathsf{VS})$;
- $\mathsf{LFWL}(2) \nprec \mathsf{SWL}(\mathsf{SV})$, $\mathsf{LFWL}(2) \nprec \mathsf{PSWL}(\mathsf{SV})$, $\mathsf{LFWL}(2) \nprec \mathsf{GSWL}$, $\mathsf{LFWL}(2) \nprec \mathsf{SSWL}$.

Due to space limitations, we can only present a brief proof sketch below, but we strongly encourage readers to browse the proof in Appendix I, where novel counterexamples for all these cases are constructed and analyzed using the pebbling game developed in Section 6. This is highly non-trivial and is a major technical contribution of this paper.

Our counterexamples are motivated by Fürer (2001). Given a base graph F, Fürer (2001) gave a principled way to construct a pair of non-isomorphic but highly similar graphs

$G(F)$ and $H(F)$ that cannot be distinguished by k-FWL. The key insight is that the difference between $G(F)$ and $H(F)$ is caused by a "twist" operation. (One can imagine the two graphs as a circle strip and its corresponding Möbius strip.) To distinguish the two graphs, Spoiler's only strategy is to fence out a twisted edge using his pebbles, similar to the strategy in Go. Yet, their analysis only applies to k-FWL algorithms. We considerably generalize Fürer's approach by noting that different SWL/FWL-type algorithms differ significantly in their "surrounding" capability in the pebbling game. Given two WL algorithms $\mathsf{A}_1, \mathsf{A}_2$ where we want to prove $\mathsf{A}_1 \prec \mathsf{A}_2$, we can identify the extra surrounding capability of A_2 and skillfully construct a base graph F such that the extra power is *necessary* to fence out a twisted edge. Here, the main challenge lies in constructing base graphs, which are given in Figures 4 to 11.

In Figure 1, we give a clear illustration of the relationships between different SWL/FWL-type algorithms stated in Theorem 7.1, which forms a complete and elegant hierarchy. In the next section, we will give a detailed discussion of the significance of Theorem 7.1 in the context of prior works.

8. Discussions with Prior Works

The theoretical results in this paper can be directly used to analyze and compare the expressiveness of various subgraph GNNs in prior work, as summarized below:

Proposition 8.1. *Under the node marking policy,* (i) ReconstructionGNN *(Cotta et al., 2021),* NGNN *(Zhang & Li, 2021),* IDGNN *(You et al., 2021), and* DS-GNN *(Bevilacqua et al., 2022) are as expressive as* SWL(VS)*;* (ii) OSAN *(Qian et al., 2022) is as expressive as* SWL(SV)*;* (iii) GNN-AK *(Zhao et al., 2022a) is as expressive as* PSWL(VS)*;* (iv) DSS-GNN *(or* ESAN*) (Bevilacqua et al., 2022),* GNN-AK-ctx *(Zhao et al., 2022a), and* SUN *(Frasca et al., 2022) are as expressive as* GSWL*;* (v) RelGN(2) *(Frasca et al., 2022) is as expressive as* SSWL.

Proof. The proof of ReconstructionGNN, NGNN, IDGNN, DS-GNN, and OSAN follows by directly using Corollary 4.7 since these subgraph GNNs fit our framework of Definition 2.1. For other architectures, the proof can be found in Appendix F. □

Regarding open problems in prior works. Below, we show how our results can be used to settle a series of open problems raised before.

In Bevilacqua et al. (2022), the authors proposed two variants of WL algorithms, the DS-WL and the DSS-WL. They conjectured that the latter is strictly more powerful than the former due to the introduced cross-graph aggregation. Very recently, Zhang et al. (2023) gave the first evidence to this conjecture by proving that DSS-WL can distinguish cut vertices using node colors while DS-WL cannot. However, since identifying cut vertices is a *node-level* task, it remains an open question when considering the standard *graph-level* expressiveness, in particular, the task of distinguishing non-isomorphic graphs. Our result fully addressed the open question by showing that DSS-WL is indeed strictly more powerful than DS-WL in distinguishing non-isomorphic graphs.

In Zhao et al. (2022a), the authors proposed two GNN architectures: GNN-AK and its extension GNN-AK-ctx. GNN-AK incorporates the so-called *centroid encoding* and GNN-AK-ctx further incorporates the *contextual encoding*. While the authors empirically showed the effectiveness of these encodings and found that GNN-AK-ctx can achieve much better performance on real-world tasks, they did not give a theoretical justification. Here, our result provides deep insights into the two models by indicating that (i) with centroid encoding, GNN-AK is strictly more powerful than vanilla subgraph GNNs; (ii) with contextual encoding, GNN-AK-ctx is strictly more powerful than GNN-AK.

Recently, Qian et al. (2022) proposed two classes of subgraph GNNs, the original OSAN and the vertex-subgraph OSAN, which differ only in the final pooling paradigm. However, the authors did not discuss the relationship between the two types of architectures. Indeed, one may naturally guess that they have the same expressive power given the same GNN backbone. However, our result highlights that it is not the case: the original 1-OSAN is strictly more powerful than vertex-subgraph 1-OSAN.

Recently, Frasca et al. (2022) proposed a theoretically-inspired model called RelGN(2), as well as a practical version called SUN that unifies prior node-based subgraph GNNs. The authors conjectured that these models are more powerful than prior architectures and may even match the power of 2-FWL. It is formally left as an important open problem to study the expressiveness lower bound of RelGN(2) and SUN (Frasca et al., 2022, Appendix E). In this paper, we fully settle the open problem by showing that: (i) RelGN(2) is indeed the strongest subgraph GNN model and is strictly more powerful than prior models; (ii) However, SUN is just as powerful as the simpler ESAN although it incorporates many extra equivariant aggregation operations; (iii) RelGN(2) *does not* achieve the 2-FWL expressiveness. Moreover, we point out an inherent gap between RelGN(2) and 2-FWL, showing that RelGN(2) even does not match SLFWL(2), a much weaker WL algorithm with the same complexity as RelGN(2).

Finally, we note that Frasca et al. (2022) mentioned two basic atomic aggregations that are not included in prior subgraph GNNs: $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$ and $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$ (see Definition 3.1). In this paper, we highlight that they are actually fundamental: incorporating either of them into the subgraph GNN layer can essentially improve the model's expressiveness.

Discussions with Morris et al. (2020). Our results also reveal a surprising relationship between the work of Morris et al. (2020) and subgraph GNNs. In Morris et al. (2020), the authors proposed the so-called δ-2-LWL, which can be seen as the symmetrized version of local 2-WL test. The update formula of δ-2-LWL is written as follows:

$$\chi_G^{(t+1)}(u,v) = \mathsf{hash}\left(\chi_G^{(t)}(u,v), \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\}, \{\!\{\chi_G^{(t)}(w,v) : w \in \mathcal{N}_G(u)\}\!\}\right).$$

δ-2-LWL shares great similarities with SSWL in the update formula. Actually, while the two algorithms differ in the initial color and the final pooling paradigm, we can prove that δ-2-LWL is *as powerful as* SSWL. We thus obtain the following key results:

- Subgraph GNNs are also bounded by δ-2-LWL. Moreover, the strongest subgraph GNN, such as $\mathsf{RelGN}(2)$, matches the power of δ-2-LWL. This builds an interesting link between the works of Frasca et al. (2022) and Morris et al. (2020).
- There is a fundamental gap between localized 2-WL and localized 2-FWL, despite the fact that both algorithms have the same computation/memory complexity. Such a result is perhaps surprising: it strongly contrasts to the relation between standard WL and FWL algorithms, where algorithms with equal computational complexity (e.g., k-FWL and $(k+1)$-WL) always have the *same* expressive power.

9. Experiments

Our theory also provides clear guidance in designing simple, efficient, yet powerful subgraph GNN architectures. In particular, we find that all the previously proposed practical node-based subgraph GNNs are bounded by GSWL (Proposition 8.1), which does not attain the maximal power in the SWL hierarchy. Instead, we decide to adopt the elegant, SSWL-based subgraph GNN design principle, resulting in only 3 atomic equivariant aggregation operations; yet the corresponding model, called GNN-SSWL, is strictly more powerful than all prior node-based subgraph GNNs. We also design an extension of GNN-SSWL, denoted as GNN-SSWL+, by further incorporating the single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$. While this does not improve the model's expressivity in theory, we find that it can often achieve better performance in real-world tasks. In addition, motivated by Proposition 4.2, the graph generation policy for both GNN-SSWL and GNN-SSWL+ is chosen as the distance encoding on the original graph (which is as expressive as node marking). A detailed description of model configuration and training hyper-parameters is given in Appendix K. Our code will be released at https://github.com/subgraph23/SWL.

Performance on Counting Substructure Benchmark. Following Zhao et al. (2022a); Frasca et al. (2022), we first consider the synthetic task of counting substructures. The result is presented in **Table 1**. Our proposed models can solve all tasks almost completely and performs better than all prior node-based subgraph GNNs on most substructures, such as triangle, tailed triangle, 4-cycle, 5-cycle, and 6-cycle. In particular, GNN-SSWL+ significantly outperform GNN-AK+ and SUN for counting 6-cycles. We suspect that GSWL is not expressive for counting 6-cycle while SSWL is expressive for this task, which may highlight a fundamental advantage of SSWL in practical scenarios when the ability to count 6-cycle is needed (Huang et al., 2022).

Performance on ZINC benchmark. We then validate our proposed models on the ZINC molecular property prediction benchmark (Dwivedi et al., 2020). We consider both ZINC-subset (12K selected graphs) and ZINC-full (250k graphs) and compare our models with both subgraph GNNs and other typical methods, such as substructure-based GNNs (Bouritsas et al., 2022; Bodnar et al., 2021a) and Graph Transformers (Zhang et al., 2023).

The result is presented in **Table 2**. *First*, it can be observed that our proposed GNN-SSWL already matches/outperforms the performance of all subgraph GNN baselines while being much simpler. In particular, compared with state-of-the-art SUN architecture, GNN-SSWL requires only a quarter of atomic aggregations in each GNN layer and roughly half of the parameters, yet matches the performance of SUN on ZINC-subset. *Second*, by further incorporating $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, GNN-SSWL+ significantly surpasses all subgraph GNN baselines and achieves state-of-the-art performance on both tasks. *Finally*, an interesting finding is that the performance of different subgraph GNN architectures shown in Table 2 roughly aligns with their theoretical expressivity in the SWL hierarchy. This may further justify that designing theoretically more powerful subgraph GNNs can benefit real-world tasks as well.

Other tasks. We also conduct experiments on the OGBG-molhiv dataset (Hu et al., 2020). Due to space limit, the result is presented in Appendix K.4.

10. Conclusion

This paper gives a comprehensive and unified analysis of the expressiveness of subgraph GNNs. By building a complete expressiveness hierarchy, one can gain deep insights into the power and limitation of various prior works. On the theoretical side, we reveal close relations between SWL, localized WL, and localized Folklore WL, and propose a unified analyzing framework via pebbling games. On the practical side, we design a simple yet powerful subgraph GNN architecture that achieves strictly better expressivity and superior performance on multiple benchmarks.

References

Arvind, V., Fuhlbrück, F., Köbler, J., and Verbitsky, O. On weisfeiler-leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113:42–59, 2020.

Azizian, W. and Lelarge, M. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

Balcilar, M., Héroux, P., Gauzere, B., Vasseur, P., Adam, S., and Honeine, P. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pp. 599–608. PMLR, 2021.

Barceló, P., Geerts, F., Reutter, J., and Ryschkov, M. Graph neural networks with local graph parameters. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25280–25293, 2021.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

Bodnar, C., Frasca, F., Otter, N., Wang, Y. G., Liò, P., Montufar, G., and Bronstein, M. M. Weisfeiler and lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021a.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lio, P., and Bronstein, M. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pp. 1026–1037. PMLR, 2021b.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Cai, J.-Y., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

Chandra, A. K., Raghavan, P., Ruzzo, W. L., Smolensky, R., and Tiwari, P. The electrical resistance of a graph captures its commute and cover times. *computational complexity*, 6(4):312–340, 1996.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 10383–10395, 2020.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13260–13271, 2020.

Cotta, L., Morris, C., and Ribeiro, B. Reconstruction for powerful graph representations. In *Advances in Neural Information Processing Systems*, volume 34, pp. 1713–1726, 2021.

Cvetkovic, D., Cvetković, D. M., Rowlinson, P., and Simic, S. *Eigenspaces of graphs*. Cambridge University Press, 1997.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Ehrenfeucht, A. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49 (129-141):13, 1961.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fraïsaé, R. Sur quelques classifications des systèmes de relations. *Publications Scientifiques de lÚniversité dÁlger*, 1954.

Frasca, F., Bevilacqua, B., Bronstein, M. M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. In *Advances in Neural Information Processing Systems*, 2022.

Fürer, M. Weisfeiler-lehman refinement requires at least a linear number of iterations. In *International Colloquium on Automata, Languages, and Programming*, pp. 322–333. Springer, 2001.

Fürer, M. On the combinatorial power of the weisfeiler-lehman algorithm. In *International Conference on Algorithms and Complexity*, pp. 260–271. Springer, 2017.

Geerts, F. and Reutter, J. L. Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, volume 30, pp. 1025–1035, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in neural information processing systems*, volume 33, pp. 22118–22133, 2020.

Huang, Y., Peng, X., Ma, J., and Zhang, M. Boosting the cycle counting power of graph neural networks with i2-gnns. *arXiv preprint arXiv:2210.13978*, 2022.

Immerman, N. and Lander, E. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pp. 59–81. Springer, 1990.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Kwon, J., Kim, J., Park, H., and Choi, I. K. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In *International Conference on Machine Learning*, pp. 5905–5914. PMLR, 2021.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: design provably more powerful neural networks for graph representation learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 4465–4478, 2020.

Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In *Advances in neural information processing systems*, volume 32, pp. 2156–2167, 2019a.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y. On the universality of invariant networks. In *International conference on machine learning*, pp. 4363–4371. PMLR, 2019c.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Morris, C., Rattan, G., and Mutzel, P. Weisfeiler and leman go sparse: towards scalable higher-order graph embeddings. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 21824–21840, 2020.

Morris, C., Rattan, G., Kiefer, S., and Ravanbakhsh, S. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning*, pp. 16017–16042. PMLR, 2022.

Papp, P. A. and Wattenhofer, R. A theoretical comparison of graph neural network extensions. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 17323–17345, 2022.

Papp, P. A., Martinkus, K., Faber, L., and Wattenhofer, R. Dropgnn: random dropouts increase the expressiveness of graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21997–22009, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Puny, O., Lim, D., Kiani, B. T., Maron, H., and Lipman, Y. Equivariant polynomials for graph neural networks. *arXiv preprint arXiv:2302.11556*, 2023.

Qian, C., Rattan, G., Geerts, F., Niepert, M., and Morris, C. Ordered subgraph aggregation networks. In *Advances in Neural Information Processing Systems*, 2022.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Vignac, C., Loukas, A., and Frossard, P. Building powerful and equivariant graph neural networks with structural message-passing. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 14143–14155, 2020.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34, 2021.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *International conference on machine learning*, pp. 7134–7143. PMLR, 2019.

You, J., Gomes-Selman, J. M., Ying, R., and Leskovec, J. Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10737–10745, 2021.

Zhang, B., Luo, S., He, D., and Wang, L. Rethinking the expressive power of gnns via graph biconnectivity. In *International Conference on Learning Representations*, 2023.

Zhang, M. and Li, P. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15734–15747, 2021.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any gnn with local structure awareness. In *International Conference on Learning Representations*, 2022a.

Zhao, L., Shah, N., and Akoglu, L. A practical, progressively-expressive GNN. In *Advances in Neural Information Processing Systems*, 2022b.

Ackowledgement

Bohang Zhang would like to thank Shengjie Luo for helpful discussions on expreriments. This work is supported by National Key R&D Program of China (2022ZD0114900) and National Science Foundation of China (NSFC62276005).

Table 1. Performance comparison of different GNN architectures on the Counting Substructure benchmark. We report the Mean Absolute Error (MAE), and use different background colors to distinguish different levels of MAE.

Model	Reference	Triangle	Tailed Tri.	Star	4-Cycle	5-Cycle	6-Cycle
PPGN	Maron et al. (2019a)	0.0089	0.0096	0.0148	0.0090	0.0137	0.0167
GNN-AK	Zhao et al. (2022a)	0.0934	0.0751	0.0168	0.0726	0.1102	0.1063
GNN-AK+	Zhao et al. (2022a)	0.0123	0.0112	0.0150	0.0126	0.0268	0.0584
SUN (EGO+)	Frasca et al. (2022)	0.0079	0.0080	**0.0064**	0.0105	0.0170	0.0550
GNN-SSWL	This paper	0.0098	0.0090	0.0089	0.0107	0.0142	0.0189
GNN-SSWL+	This paper	**0.0064**	**0.0067**	0.0078	**0.0079**	**0.0108**	**0.0154**

Table 2. Performance comparison of different subgraph GNNs on ZINC benchmark. The Mean Absolute Error (MAE) and the standard deviation are reported. We also list the WL equivalence class and the number of parameters/atomic aggregations for each model.

Model	Reference	WL	# Param.	# Agg.	ZINC Test MAE Subset	ZINC Test MAE Full
GSN	Bouritsas et al. (2022)	-	∼500k	-	0.101±0.010	-
CIN (small)	Bodnar et al. (2021a)	-	∼100k	-	0.094±0.004	0.044±0.003
Graphormer-GD	Zhang et al. (2023)	GD-WL	503k	-	0.081±0.009	0.025±0.004
NGNN	Zhang & Li (2021)	SWL(VS)	∼500k	2	0.111±0.003	0.029±0.001
GNN-AK	Zhao et al. (2022a)	PSWL(VS)	∼500k	4	0.105±0.010	-
GNN-AK+	Zhao et al. (2022a)	GSWL	∼500k	5	0.091±0.002	-
ESAN	Bevilacqua et al. (2022)	GSWL	∼100k	4	0.102±0.003	0.029±0.003
ESAN	Frasca et al. (2022)	GSWL	446k	4	0.097±0.006	0.025±0.003
SUN	Frasca et al. (2022)	GSWL	526k	12	0.083±0.003	0.024±0.003
GNN-SSWL	This paper	SSWL	274k	3	0.082±0.003	0.026±0.001
GNN-SSWL+	This paper	SSWL	387k	4	**0.070±0.005**	**0.022±0.002**

Appendix

The Appendix is organized as follows:

- In Appendix A, we complete our theoretical analysis by showing that subgraph GNNs with better expressivity can also be stronger in terms of their ability to compute fundamental graph properties, such as distance and biconnectivity.
- In Appendix B,we provide a broader review of related literature in the area of expressive GNNs, including higher-order GNNs, sparsity-aware GNNs, subgraph GNNs, and more.
- In Appendix C, we list a few promising open directions of this paper.
- In Appendix D, we give the missing proof of Proposition 3.2, showing the equivalence between SWL and Subgraph GNNs.
- In Appendix E, we give all the missing proofs in Section 4, which we use to build a complete hierarchy of SWL algorithms. This part is technical and is divided into several subsections (from Appendices E.1 to E.5).
- In Appendix F, we discuss several subgraph GNNs beyond our proposed framework (Definition 2.1), including GNN-AK, GNN-AK-ctx, ESAN, SUN, and RelGN(2). We show that each of these architectures still corresponds to an equivalent SWL algorithm in terms of expressive power.
- In Appendix G, we give the missing proof of Theorem 5.2, showing the expressivity relationships between different SWL and localized FWL algorithms.
- In Appendix H, we give all the missing proofs in Section 6, bridging SWL/FWL-type algorithms and pebbling games.
- In Appendix I, we give the missing proof of Theorem 7.1. The proof is non-trivial and contains the main technical contribution of this paper. It is divided into three parts in Appendices I.1 to I.3 for readability.
- In Appendix J, we give all the missing proofs in Appendix A, showing how various SWL algorithms differ in terms of their practical expressiveness such as encoding graph distance and biconnectivity.
- In Appendix K, we provide experimental details to reproduce the results in Section 9, as well as a comprehensive set of ablation studies.

A. Discussions on Practical Expressiveness

Up to now, we have obtained precise expressivity relations for all pairs of SWL/FWL-type algorithms in distinguishing non-isomorphic graphs. From a practical perspective, however, one may still wonder whether/how GNNs designed based on a theoretically stronger WL algorithm can be more powerful in solving *practical* graph problems. Here, we give concrete evidence that the power of different SWL algorithms does vary in terms of computing fundamental graph properties. In particular, WL algorithms with expressiveness over PSWL are capable of encoding *distance* and *biconnectivity* of a graph, while weaker algorithms like the vanilla SWL are unable to fully encode any of them.

Our result is motivated by the recent study of Zhang et al. (2023), who proposed a new class of WL algorithms called the Generalized Distance WL (GD-WL). Given graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, GD-WL maintains a color $\chi_G(v)$ for each node $v \in \mathcal{V}_G$, and the node color is updated according to the following formula:

$$\chi_G^{(t+1)}(v) := \mathsf{hash}(\{\!\{(d_G(u,v), \chi_G^{(t)}(u)) : u \in \mathcal{V}\}\!\}),$$

where $d_G(u,v)$ is a generalized distance between u and v. Zhang et al. (2023) proved that, by incorporating both the *shortest path distance* (SPD) and the *resistance distance* (RD), i.e., setting $d_G(u,v) = (\mathrm{dis}_G(u,v), \mathrm{dis}_G^{\mathsf{R}}(u,v))$, the resulting GD-WL is provably expressive for all types of biconnectivity metrics, such as identifying cut vertices, cut edges, or distinguishing non-isomorphic graphs with different block cut trees. Surprisingly, we find that the PSWL has intrinsically (implicitly) encoded another type of GD-WL defined as follows:

Definition A.1 (Hitting time distance)**.** Define $\mathrm{dis}_G^{\mathsf{H}}(u,v)$ to be the hitting time distance (HTD) from node u to v in graph G, i.e., the average number of edges passed in a random walk starting from u and reaching v for the first time.

Theorem A.2. *Let* $d_G(u,v) = (\mathrm{dis}_G(u,v), \mathrm{dis}_G^{\mathsf{H}}(u,v))$. *Then,* GD-WL $\prec$ PSWL(VS).

Hitting time distance is closely related to resistance distance, in that $\mathrm{dis}_G^{\mathsf{R}}(u,v) = (\mathrm{dis}_G^{\mathsf{H}}(u,v) + \mathrm{dis}_G^{\mathsf{H}}(v,u))/2|\mathcal{E}_G|$ holds for any graph G and nodes $u, v \in \mathcal{V}_G$ (Chandra et al., 1996). In other words, RD can be seen as the symmetrized version of HTD (by ignoring the constant $1/|\mathcal{E}_G|$). Moreover, we have the following theorem, showing that HTD-WL also resembles RD-WL in distinguishing vertex-biconnectivity:

Theorem A.3. *By setting* $d_G = \mathrm{dis}_G^{\mathsf{H}}$*, the resulting HTD-WL is fully expressive for all vertex-biconnectivity metrics proposed in Zhang et al. (2023).*

The proofs of Theorems A.2 and A.3 are given in Appendix J. Combining the two theorems readily leads to the following corollary:

Corollary A.4. PSWL(VS) *is fully expressive for both edge-biconnectivity and vertex-biconnectivity.*

On the other hand, we find that the vanilla SWL is unable to fully encode either SPD, HTD, or RD, as shown in the proposition below:

Proposition A.5. *The following hold:*

- SWL(VS) $\nprec$ SPD-WL, SWL(SV) $\nprec$ SPD-WL*;*
- SWL(VS) $\nprec$ HTD-WL, SWL(SV) $\nprec$ HTD-WL*;*
- SWL(VS) $\nprec$ RD-WL, SWL(SV) $\nprec$ RD-WL.

Besides, Zhang et al. (2023) has shown that the SWL(VS) cannot identify cut vertices of a graph. Therefore, incorporating extra aggregation operations in the vanilla SWL *does* essentially improve its practical expressiveness in computing basic graph properties like distance and biconnectivity.

A further discussions with Zhang et al. (2023). In Zhang et al. (2023), the authors showed that most prior GNN models are not expressive for biconnectivity metrics except ESAN (Bevilacqua et al., 2022), which corresponds to GSWL in our framework. Here, we unify, justify, and extend their results/findings in the following aspects:

- We show ESAN can identify cut vertices mainly because it encodes the generalized distance. This provides deep insights into ESAN and complements the finding that ESAN can encode SPD. From this perspective, we obtain an alternative and unified proof for ESAN in distinguishing vertex-biconnectivity. Moreover, we also prove that ESAN can distinguish the block cut-vertex tree, a new result that was not originally proved in Zhang et al. (2023).

- We strongly justify the introduced generalized distance and GD-WL as a fundamental class of color refinement algorithms, since the reason why ESAN and other SWL variants can encode biconnectivity metrics simply lies in the fact that it is more powerful than GD-WL.
- In contrast, we prove that the weaker SWL(VS) (or SWL(SV)) is not more powerful than either SPD-WL or RD-WL. This explains and complements the finding in Zhang et al. (2023) on why DS-WL cannot identify cut vertices. We also partially answered the question in Zhang et al. (2023) for OS-WL (Qian et al., 2022).
- We show adding the global aggregation in vanilla SWL (like ESAN) is not the only way to make it expressive for biconnectivity metrics. In particular, simply adding a single-point aggregation (PSWL) already suffices.

Remark A.6. We suspect that the PSWL can also encode the resistance distance, but currently we can only prove that the strongest SSWL can encode RD (Appendix J). We leave this as an open problem for future work.

B. Related Work

Since Xu et al. (2019); Morris et al. (2019) discovered the limited expressiveness of vanilla MPNNs, a large amount of works have been devoted to developing GNNs with better expressive power. Here, we briefly review the literature on expressive GNNs that are most relevant to this paper.

Higher-order GNNs. Maron et al. (2019a;c); Azizian & Lelarge (2021); Geerts & Reutter (2022) theoretically studied the question of designing provably expressive equivariant GNNs that match the power of k-FWL test for $k > 1$. In this way, they build a hierarchy of GNNs with strictly growing expressivity (similar to this paper). A representative higher-order GNN architecture is called the k-IGN (Maron et al., 2019c): it stores a feature representation for each node k-tuple and updates these features using higher-order equivariant layers developed in Maron et al. (2019b). Recently, Frasca et al. (2022) proved that all node-based subgraph GNNs can be implemented by 3-IGN, which then implies that subgraph GNNs' expressive power is intrinsically bounded by 2-FWL (Geerts & Reutter, 2022).

Sparsity-aware GNNs. One major drawback of higher-order GNNs is that the architectural design does not well-exploit the graph structural information, since the graph adjacency is only encoded in the initial node features. In light of this, subsequent works like Morris et al. (2020; 2022); Zhao et al. (2022b) incorporated this inductive bias directly into the network layers and designed *local* versions of higher-order GNNs. For example, Morris et al. (2020) developed the so-called δ-k-LWL, which can be seen as a localized version of k-WL. Morris et al. (2022) proposed the (k, s)-SpeqNets by considering only k-tuples whose vertices can be grouped into no more than s connected components. Zhao et al. (2022b) concurrently proposed the (k, s)-SETGNN which is similar to (k, s)-SpeqNets. In this paper, we propose a class of localized k-FWL, which shares interesting similarities to δ-k-LWL. Our major contribution is to establish complete relations between localized 2-FWL, δ-2-LWL, and subgraph GNNs.

Subgraph GNNs. Subgraph GNNs are an emerging class of higher-order GNNs that compute a feature representation for each subgraph-node pair. The earliest idea of subgraph GNNs may track back to Cotta et al. (2021); Papp et al. (2021), which proposed to use node-deleted subgraphs and performed message-passing on each subgraph separately without cross-graph interaction. Papp & Wattenhofer (2022) argued to use node marking instead of node deletion for better expressive power. Zhang & Li (2021) proposed the Nested GNN (NGNN), a variant of subgraph GNNs that use k-hop ego nets with distance encoding. It further added the global aggregation $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$ to merge all node information in a subgraph when computing the feature of the root node of a subgraph. You et al. (2021) designed the ID-GNN, which is similar to NGNN and also uses k-hop ego nets as subgraphs. Bevilacqua et al. (2022) developed a principled class of subgraph GNNs, called ESAN, which first introduced the cross-graph global aggregation into the network design. Zhao et al. (2022a) concurrently proposed the GNN-AK and its extension GNN-AK-ctx, which also includes the cross-graph global aggregation. Recently, Frasca et al. (2022); Qian et al. (2022) first provided theoretical analysis of various node-based subgraph GNNs by proving that they are intrinsically bounded by 2-FWL. We note that besides node-based subgraph GNNs, one can also develop edge-based subgraph GNNs, which have been explored in Bevilacqua et al. (2022); Huang et al. (2022). Both works showed that the expressive power of edge-based subgraph GNNs can go beyond 2-FWL. Finally, we note that Vignac et al. (2020) proposed a GNN architecture that is somewhat *similar* to the vanilla subgraph GNN, and the δ-2-LWL proposed in Morris et al. (2020) can also be seen as a subgraph GNN according to Section 8.

Practical expressivity of GNNs. Another line of works sought to develop expressive GNNs from practical consideration. For example, Fürer (2017); Chen et al. (2020); Arvind et al. (2020) studied the power of WL algorithms in counting graph substructures and pointed out that vanilla MPNNs cannot count/detect cycles, which may severely limit their practical

performance in real-world tasks (e.g., in bio-chemistry). In light of this, Bouritsas et al. (2022); Barceló et al. (2021) proposed to incorporate substructure counting (or homomorphism counting) into the initial node features to boost the expressiveness. Bodnar et al. (2021b;a) further proposed a message-passing framework that enables interaction between nodes, edges, and higher-order substructures. Huang et al. (2022) studied the cycle counting power of subgraph GNNs and proposed the I^2-GNN to count cycles of length no more than 6. Recently, Puny et al. (2023) studied the expressive power of GNNs in expressing/approximating equivariant graph polynomials. They showed that computing equivariant polynomials generalizes the problem of counting substructures.

Besides cycle counting, several works explored other aspects of encoding basic graph properties. You et al. (2019); Li et al. (2020); Ying et al. (2021) proposed to use distance encoding to boosting the expressiveness of MPNNs or Graph Transformers. In particular, Li et al. (2020) proposed to use a generalized distance called page-rank distance. Balcilar et al. (2021); Kreuzer et al. (2021); Lim et al. (2022) studies the expressive power of GNNs from the perspective of graph spectral (Cvetkovic et al., 1997). Recently, Zhang et al. (2023) discovered that most prior GNN architectures are not expressive for graph biconnectivity and built an interesting relation between biconnectivity and generalized distance. Here, we extend Zhang et al. (2023) by giving a comprehensive characterization of which SWL equivalence class can encode distance and biconnectivity.

C. Open directions

We highlight several open directions for future work as follows.

Regarding higher-order subgraph GNNs. From a theoretical perspective, it is an interesting direction to generalize the results of this paper to *higher-order* subgraph GNNs (which compute a feature representation for each node k-tuple). We note that such an idea has appeared in Cotta et al. (2021); Qian et al. (2022); Papp & Wattenhofer (2022). However, none of these works explored the possible design space of *cross-graph* aggregations. Since our results imply that these cross-graph aggregations do essentially improve the expressive power, it may be worthwhile to establish a complete hierarchy of higher-order subgraph GNNs. This may include the following questions: (i) How many expressivity equivalence classes are there? (ii) What are the expressivity inclusion relations between different equivalence classes? (iii) *What design principle achieves the maximal expressive power with a minimal number of atomic aggregations?* We conjecture that, by symmetrically incorporating k local aggregations, the resulting k-order subgraph GNN achieves the maximal expressiveness and is as expressive as $\mathsf{RelGN}(k)$ (by extending Frasca et al. (2022)).

Regarding edge-based subgraph GNNs. Another different perspective is to study edge-based subgraph GNNs, which compute a feature representation for each edge-node pair. Importantly, edge-based subgraph GNNs and the corresponding SWL are also a fundamental class of computation models with $O(nm)$ memory complexity and $O(m^2)$ computation complexity. For sparse graphs (i.e. $m = O(n)$), such a complexity is quite desirable and is close to that of node-based subgraph GNNs. Yet, it results in enhanced expressiveness: as shown in Bevilacqua et al. (2022); Huang et al. (2022), their proposed edge-based subgraph GNNs are not less powerful than 2-FWL. Therefore, we believe that characterizing the expressiveness hierarchy of edge-based subgraph GNNs is of both theoretical and practical interest. Another interesting topic is to build expressivity relations between node-based and edge-based subgraph GNNs.

Regarding localized Folklore WL tests. This paper proposed a novel class of color refinement algorithms called localized Folklore WL. Importantly, we show $\mathsf{SLFWL}(2)$ is strictly more powerful than all node-based subgraph GNNs despite the same complexity. Therefore, an interest question is whether we can design practical GNN architectures based on $\mathsf{SLFWL}(2)$ for both efficiency and better expressiveness. On the other hand, from a theoretical side, it may also be an interesting direction to study higher-order localized Folklore WL tests, in particular, $\mathsf{SLFWL}(k)$, due to its fundamental nature. We conjecture that $\mathsf{SLFWL}(k)$ is strictly more powerful than δ-k-LWL (Morris et al., 2020) and strictly less powerful than standard k-FWL. Furthermore, *does* $\mathsf{SLFWL}(k)$ *achieve the maximal expressive power among the algorithm class within* $O(n^{k-1}m)$ *computation cost?*

Regarding practical expressiveness of GSWL **and** SSWL. This paper discusses the practical expressiveness of subgraph GNNs by showing an inherent gap between SWL and PSWL in terms of their ability to encode distance and biconnectivity of a graph. Yet, it remains an open problem how PSWL, GSWL, and SSWL differ in terms of their practical expressiveness for computing graph properties. This question is particularly important since recently proposed subgraph GNNs are typically bounded by GSWL. Answering this question will thus highlight the power and limitation of prior architectures.

D. The Equivalence between SWL and Subgraph GNNs

This section aims to prove Proposition 3.2. We restate the proposition below:

Proposition 3.2. *The expressive power of any subgraph GNN defined in Section 2 is bounded by a corresponding SWL by matching the policy π, the aggregation scheme between Definitions 2.1 and 3.1, and the pooling paradigm. Moreover, when considering bounded-size graphs, for any SWL algorithm, there exists a matching subgraph GNN with the same expressive power.*

Proof. We first prove that any subgraph GNN defined in Section 2 is bounded by a corresponding SWL. To do this, we will prove the following result: given a pair of graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$, for any $t \in \mathbb{N}$ and any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$, $\chi_G^{(t)}(u, v) = \chi_H^{(t)}(x, y) \implies h_G^{(t)}(u, v) = h_H^{(t)}(x, y)$, where χ and h are defined in Definitions 2.1 and 3.1, respectively.

We prove the result by induction over t. For the base case of $t = 0$, the result clearly holds when the graph generation policy is the same between SWL and subgraph GNNs. Now assume the result holds for all $t \leq T$, and we want to prove that it also holds for $t = T + 1$. By Definition 3.1, $\chi_G^{(T+1)}(u, v) = \chi_H^{(T+1)}(x, y)$ is equivalent to

$$\mathsf{agg}_i(u, v, G, \chi_G^{(T)}) = \mathsf{agg}_i(x, y, H, \chi_H^{(T)}), \quad \forall i \in [r].$$

We separately consider each type of aggregation operation:

- Single-point aggregation. Take $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$ for example: $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}(u, v, G, \chi_G^{(T)}) = \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}(x, y, H, \chi_H^{(T)})$ implies $\chi_G^{(T)}(v, u) = \chi_H^{(T)}(y, x)$. By induction, we have $h_G^{(t)}(v, u) = h_H^{(t)}(y, x)$.
- Local aggregation. Take $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ for example: $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}(u, v, G, \chi_G^{(T)}) = \mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}(x, y, H, \chi_H^{(T)})$ implies

  $$\{\!\{\chi_G^{(T)}(u, w) : w \in \mathcal{N}_{G^u}(v)\}\!\} = \{\!\{\chi_H^{(T)}(x, z) : z \in \mathcal{N}_{H^x}(y)\}\!\}.$$

  By induction, it is straightforward to see that

  $$\{\!\{h_G^{(T)}(u, w) : w \in \mathcal{N}_{G^u}(v)\}\!\} = \{\!\{h_H^{(T)}(x, z) : z \in \mathcal{N}_{H^x}(y)\}\!\}.$$

  Therefore,

  $$\sum_{w \in \mathcal{N}_{G^u}(v)} h_G^{(T)}(u, w) = \sum_{z \in \mathcal{N}_{H^x}(y)} h_H^{(T)}(x, z).$$

- Global aggregation. This case is similar to the above one and we omit it for clarity.

Combining all these cases, we have

$$\mathsf{op}_i(u, v, G, \chi_G^{(T)}) = \mathsf{op}_i(x, y, H, \chi_H^{(T)}) \quad \forall i \in [r],$$

and thus $h_G^{(T+1)}(u, v) = h_H^{(T+1)}(x, y)$. We have completed the induction step.

Let L be the number of layers in a subgraph GNN. Then $\chi_G^{(L)}(u, v) = \chi_H^{(L)}(x, y)$ implies $h_G^{(L)}(u, v) = h_H^{(L)}(x, y)$. Since $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ is always present, the stable color mapping χ satisfies that $\chi_G(u, v) = \chi_H(x, y) \implies \chi_G^{(L)}(u, v) = \chi_H^{(L)}(x, y)$. Therefore, $\chi_G(u, v) = \chi_H(x, y) \implies h_G^{(L)}(u, v) = h_H^{(L)}(x, y)$.

Finally consider the pooling paradigm. As in the analysis of global aggregation, it can be concluded that $c(G) = c(H)$ implies $f(G) = f(H)$ where $c(G)$ and $f(G)$ represent the graph representation computed by SWL and subgraph GNN, respectively. We have finished the first part of the proof.

It remains to prove that for any SWL algorithm, there exists a matching subgraph GNN with the same expressive power. The key idea is to ensure that whenever $\chi_G^{(t)}(u, v) \neq \chi_H^{(t)}(x, y)$, we have $h_G^{(t)}(u, v) \neq h_H^{(t)}(x, y)$. To achieve this, we rely on injective functions that take a set as input. When assuming that the size of the set is bounded, the injective property can be easily constructed using the approach proposed in Maron et al. (2019a), called the power-sum multi-symmetric polynomials (PMP). We note that while Maron et al. (2019a) only focused on the case when the input belongs to sets of a *fixed* size, it

can be easily extended to our case for sets of different but bounded sizes by padding zero-elements. The *summation* in PMP just coincides with the aggregation in Definition 2.1, and the power can be extracted by the function $\sigma^{(t)}$ in the previous layer. For more details, please refer to Maron et al. (2019a).

Finally, note that when the input graph has bounded size N, the SWL iteration must get stabled in no more than N^2 steps. Therefore, by using a sufficiently deep GNN (i.e., $L = N^2$), one can guarantee that $\chi_G(u,v) \neq \chi_H(x,y)$ implies $h_G^{(L)}(u,v) \neq h_H^{(L)}(x,y)$. This eventually yields that $c(G) \neq c(H)$ implies $f(G) \neq f(H)$, as desired. □

E. Proof of Theorems in Section 4

This section contains all the missing proofs in Section 4.

E.1. Preliminary

We first introduce some basic terminologies and facts, which will be frequently used in subsequent proofs.

Definition E.1. Let χ and $\tilde{\chi}$ be two color mappings, with $\chi_G(u,v)$ and $\tilde{\chi}_G(u,v)$ representing the color of vertex pair (u,v) in graph G. We say:

- $\tilde{\chi}$ is *finer* than χ, denoted as $\tilde{\chi} \preceq \chi$, if for any two graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$, $H = (\mathcal{V}_H, \mathcal{E}_H)$ and any vertices $u, v \in \mathcal{V}_G$, $x, y \in \mathcal{V}_H$, we have $\tilde{\chi}_G(u,v) = \tilde{\chi}_H(x,y) \implies \chi_G(u,v) = \chi_H(x,y)$.
- $\tilde{\chi}$ and χ are equivalent, denoted as $\tilde{\chi} \simeq \chi$, if $\tilde{\chi} \preceq \chi$ and $\chi \preceq \tilde{\chi}$.
- $\tilde{\chi}$ is *strictly finer* than χ, denoted as $\tilde{\chi} \prec \chi$, if $\tilde{\chi} \preceq \chi$ and $\tilde{\chi} \not\simeq \chi$.

Remark E.2. Several simple facts regarding this definition are as follows.

(a) For any color refinement algorithm, let $\{\chi^{(t)}\}_{t=0}^{\infty}$ be the sequence of color mappings generated at each iteration t, then $\chi^{(t+1)} \preceq \chi^{(t)}$ for any t. This is exactly why we call the algorithm "color refinement". As a result, the stable color mapping χ is finer than any intermediate color mapping $\chi^{(t)}$.

(b) Definition E.1 is closely related to the power of WL algorithms. Indeed, let χ^{A} and χ^{B} be two stable color mappings generated by algorithms A and B, respectively. If both algorithms use the same pooling paradigm, then $\chi^{\mathsf{B}} \preceq \chi^{\mathsf{A}}$ implies that B is more powerful than A, i.e. $\mathsf{A} \preceq \mathsf{B}$.

(c) Consider two SWL algorithms A and B with the same graph generation policy, but with different aggregation schemes $\mathcal{A}$ and $\mathcal{B}$. Denote χ^{A} and χ^{B} as the corresponding stable color mappings. Define a new color mapping $\tilde{\chi} = \mathscr{T}(\mathcal{A}, \chi^{\mathsf{B}})$ that "refines" χ^{B} using aggregation scheme $\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{uv}}^{\mathsf{P}}\}$:

$$[\mathscr{T}(\mathcal{A}, \chi)]_G(u,v) = \mathsf{hash}(\mathsf{agg}_1(u,v,G,\chi_G), \cdots, \mathsf{agg}_r(u,v,G,\chi_G)), \tag{3}$$

where $\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{uv}}^{\mathsf{P}}\} = \{\mathsf{agg}_i : i \in [r]\}$. Then we can prove that $\chi^{\mathsf{B}} \preceq \tilde{\chi} \implies \chi^{\mathsf{B}} \preceq \chi^{\mathsf{A}}$. If the two algorithms further share the same pooling paradigm, Remark E.2(b) yields $\mathsf{A} \preceq \mathsf{B}$. This gives a simple way to compare the expressiveness of different algorithms.

Proof of Remark E.2(c). Define a sequence of color mappings $\{\tilde{\chi}^{(t)}\}_{t=0}^{\infty}$ recursively, such that $\tilde{\chi}^{(0)} = \chi^{\mathsf{B}}$ and

$$\tilde{\chi}_G^{(t+1)}(u,v) = \mathsf{hash}(\mathsf{agg}_1(u,v,G,\tilde{\chi}_G^{(t)}), \cdots, \mathsf{agg}_r(u,v,G,\tilde{\chi}_G^{(t)}))$$

for any u, v in graph G. Clearly, $\tilde{\chi}^{(1)}$ is just $\tilde{\chi}$ in Remark E.2(c). Since we have both $\tilde{\chi}^{(0)} \preceq \tilde{\chi}^{(1)}$ (by the assumption of $\chi^{\mathsf{B}} \preceq \tilde{\chi}$) and $\tilde{\chi}^{(1)} \preceq \tilde{\chi}^{(0)}$ (by Remark E.2(a)), $\tilde{\chi}^{(1)} \simeq \tilde{\chi}^{(0)}$. Therefore, $\tilde{\chi}^{(t)} \simeq \chi^{\mathsf{B}}$ holds for all $t \in \mathbb{N}$. On the other hand, a simple induction over t yields $\tilde{\chi}^{(t)} \preceq \chi^{\mathsf{A},(t)}$ (where $\chi^{\mathsf{A},(t)}$ is the color mapping at iteration t for algorithm A), since they are both refined by the same aggregation scheme $\mathcal{A}$ (for the base case, $\tilde{\chi}^{(0)} = \chi^{\mathsf{B}} \preceq \chi^{\mathsf{B},(0)} = \chi^{\mathsf{A},(0)}$). By taking $t \to \infty$, this finally yields $\tilde{\chi} \preceq \chi^{\mathsf{A}}$, namely, $\chi^{\mathsf{B}} \preceq \chi^{\mathsf{A}}$, as desired. □

E.2. Discussions on graph generation policies

In this subsection, we make a detailed discussion regarding graph generalization policies and prove that the canonical node marking policy already achieves the best expressiveness among all of these policies (Proposition 4.2).

Depending on the choice of (G^u, h_G^u), there are a total of 7 non-trivial combinations. For ease of presentation, we first give symbols to each of them:

- NM: the node marking policy on the original graph;
- DE: the distance encoding policy on the original graph;
- $\mathsf{EGO}(k)$: the k-hop ego network policy with constant node features;
- $\mathsf{EGO}(k)$+NM: the policy with both node marking and k-hop ego network;
- $\mathsf{EGO}(k)$+DE: the policy with both distance encoding and k-hop ego network;
- ND: the node deletion policy with constant node features;
- NDM: the policy with both node deletion and marking.

Proposition E.3. *Consider any fixed aggregation scheme $\mathcal{A}$ that contains the two basic aggregations $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ and $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ in Definition 3.1, and consider any fixed pooling paradigm Pool defined in Section 3. We have the following result:*

- NM *is as powerful as* DE*;*
- $\mathsf{EGO}(k)$+NM *is as powerful as* $\mathsf{EGO}(k)$+DE*;*
- NM *is more powerful than* $\mathsf{EGO}(k)$+NM*;*
- NM *is more powerful than* NDM*;*
- $\mathsf{EGO}(k)$+NM *is more powerful than* $\mathsf{EGO}(k)$*;*
- NDM *is more powerful than* ND.

Proof. Let $\chi^{\mathsf{Policy},(t)}$ be the color mapping of the SWL algorithm with graph generation policy Pocily, aggregation scheme $\mathcal{A}$, and pooling paradigm Pool at iteration t, and let χ^{Policy} be the corresponding stable color mapping. Here, $\mathsf{Pocily} \in \{\mathsf{NM}, \mathsf{DE}, \mathsf{EGO}(k), \mathsf{EGO}(k)\text{+}\mathsf{NM}, \mathsf{EGO}(k)\text{+}\mathsf{DE}, \mathsf{ND}, \mathsf{NDM}\}$.

We first consider the case NM vs. DE. By definition, $\chi^{\mathsf{DE},(0)}$ is finer than $\chi^{\mathsf{NM},(0)}$. Since the subgraphs $\{\!\{G^u : u \in \mathcal{V}_G\}\!\}$ are the same for the two policies and the aggregation scheme $\mathcal{A}$ is fixed, a simple induction over t then yields $\chi^{\mathsf{DE},(t)} \preceq \chi^{\mathsf{NM},(t)}$ for any $t \in \mathbb{N}$, namely, $\chi^{\mathsf{DE}} \preceq \chi^{\mathsf{NM}}$. It follows that DE is more powerful than NM (by Remark E.2(b)).

To prove the converse direction, we leverage Lemma E.4 (which will be proved later). Lemma E.4 implies that $\chi^{\mathsf{NM},(D)} \preceq \chi^{\mathsf{DE},(0)}$ when the input graphs have bounded diameter D. Then using the same analysis as above, we have $\chi^{\mathsf{NM},(D+t)} \preceq \chi^{\mathsf{DE},(t)}$ for any $t \in \mathbb{N}$. By taking $t \to \infty$, this implies that $\chi^{\mathsf{NM}} \preceq \chi^{\mathsf{DE}}$, namely, NM is more powerful than DE (by Remark E.2(b)). Combining the two directions concludes the proof of the first bullet.

The proof for the case $\mathsf{EGO}(k)$+NM vs. $\mathsf{EGO}(k)$+DE is almost the same, so we omit it for clarity.

We next turn to the case NM vs. $\mathsf{EGO}(k)$+NM. Initially, by definition we have $\chi^{\mathsf{NM},(0)} = \chi^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(0)}$. Therefore, $\chi^{\mathsf{NM},(D)} \preceq \chi^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(0)}$ (by Remark E.2(a)), where we assume the input graphs have bounded diameter D. Below, we aim to prove that $\chi^{\mathsf{NM},(t)} \preceq \chi^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(t-D)}$ for any integer $t \geq D$. We prove it by induction.

The base case of $t = D$ already holds. Assume the above result holds for $t = T$ and consider $t = T + 1$. Let $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ be two graphs with diameter no more than D. Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi_G^{\mathsf{NM},(T+1)}(u, v) = \chi_H^{\mathsf{NM},(T+1)}(x, y)$, and we want to prove that $\chi_G^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(T+1-D)}(u, v) = \chi_H^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(T+1-D)}(x, y)$. By Definition 3.1,

$$\mathsf{agg}_i(u, v, G, \chi_G^{\mathsf{NM},(T)}) = \mathsf{agg}_i(x, y, H, \chi_H^{\mathsf{NM},(T)})$$

holds for all $i \in [r]$. If agg_i is any single-point aggregation or global aggregation, by induction we clearly have $\mathsf{agg}_i(u, v, G, \chi_G^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(T-D)}) = \mathsf{agg}_i(x, y, H, \chi_H^{\mathsf{EGO}(k)\text{+}\mathsf{NM},(T-D)})$. If agg_i is any local aggregation, e.g., $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, we have

$$\{\!\{\chi_G^{\mathsf{NM},(T)}(u, w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi_H^{\mathsf{NM},(T)}(x, z) : z \in \mathcal{N}_H(y)\}\!\} \tag{4}$$

for policy NM. We additionally need to prove that

$$\{\!\{\chi_G^{\mathsf{NM},(T)}(u, w) : w \in \mathcal{N}_{G^u}(v)\}\!\} = \{\!\{\chi_H^{\mathsf{NM},(T)}(x, z) : z \in \mathcal{N}_{H^x}(y)\}\!\}, \tag{5}$$

where G^u and H^x are generated by policy $\mathsf{EGO}(k)$+NM. This is due to the following observations: if $w \in \mathcal{N}_G^k(u)$ and $z \notin \mathcal{N}_H^k(x)$, then $\mathrm{dis}_G(u, w) \neq \mathrm{dis}_H(x, z)$. Therefore, by Lemma E.4 we have $\chi_G^{\mathsf{NM},(T)}(u, w) \neq \chi_H^{\mathsf{NM},(T)}(x, z)$. Similarly,

if $w \notin \mathcal{N}_G^k(u)$ and $z \in \mathcal{N}_H^k(x)$, then $\chi_G^{\mathsf{NM},(T)}(u,w) \neq \chi_H^{\mathsf{NM},(T)}(x,z)$. This yields (5). By induction,

$$\{\!\{\chi_G^{\mathsf{EGO}(k)+\mathsf{NM},(T-D)}(u,w) : w \in \mathcal{N}_{G^u}(v)\}\!\} = \{\!\{\chi_H^{\mathsf{EGO}(k)+\mathsf{NM},(T-D)}(x,z) : z \in \mathcal{N}_{H^x}(y)\}\!\}.$$

Therefore, in all cases we have

$$\mathsf{agg}_i(u,v,G,\chi_G^{\mathsf{EGO}(k)+\mathsf{NM},(T-D)}) = \mathsf{agg}_i(x,y,H,\chi_H^{\mathsf{EGO}(k)+\mathsf{NM},(T-D)}).$$

This concludes the induction step. We finally obtain that the stable mappings satisfy $\chi^{\mathsf{NM}} \preceq \chi^{\mathsf{EGO}(k)+\mathsf{NM}}$ and thus NM is more powerful than $\mathsf{EGO}(k)$+NM (by Remark E.2(b)).

We next turn to the case NM vs. NDM. This case is similar to the above one. Initially, by definition we have $\chi^{\mathsf{NM},(0)} = \chi^{\mathsf{NDM},(0)}$. Therefore, $\chi^{\mathsf{NM},(D)} \preceq \chi^{\mathsf{NDM},(0)}$ where we assume the input graphs have bounded diameter D. We aim to prove that $\chi^{\mathsf{NM},(t)} \preceq \chi^{\mathsf{NDM},(t-D)}$ for any integer $t \geq D$. We prove it by induction. The base case of $t = D$ already holds.

Assume the above result holds for $t = T$ and consider $t = T+1$. Let $\chi_G^{\mathsf{NM},(T+1)}(u,v) = \chi_H^{\mathsf{NM},(T+1)}(x,y)$. Then by Definition 3.1,

$$\mathsf{agg}_i(u,v,G,\chi_G^{\mathsf{NM},(T)}) = \mathsf{agg}_i(x,y,H,\chi_H^{\mathsf{NM},(T)})$$

holds for all $i \in [r]$. If agg_i is any single-point aggregation or global aggregation, by induction we have $\mathsf{agg}_i(u,v,G,\chi_G^{\mathsf{NDM},(T-D)}) = \mathsf{agg}_i(x,y,H,\chi_H^{\mathsf{NDM},(T-D)})$. If agg_i is any local aggregation, e.g., $\mathsf{agg}_\mathsf{u}^\mathsf{L}$, we have (4) for policy NM, and we additionally need to prove (5), where G^u and H^x are generated by policy NDM. Note that we have $\mathrm{dis}_G(u,v) = \mathrm{dis}_H(x,y)$ due to the assumption $\chi_G^{\mathsf{NM},(T+1)}(u,v) = \chi_H^{\mathsf{NM},(T+1)}(x,y)$ and Lemma E.4. Consider the following three cases:

- If $\mathrm{dis}_G(u,v) = \mathrm{dis}_H(x,y) \geq 2$, then $\mathcal{N}_{G^u}(v) = \mathcal{N}_G(v)$ and $\mathcal{N}_{H^x}(y) = \mathcal{N}_H(y)$.
- If $\mathrm{dis}_G(u,v) = \mathrm{dis}_H(x,y) = 1$, then $\mathcal{N}_{G^u}(v) = \mathcal{N}_G(v)\backslash\{u\}$ and $\mathcal{N}_{H^x}(y) = \mathcal{N}_H(y)\backslash\{x\}$. We also have $\chi_G^{\mathsf{NM},(T)}(u,u) = \chi_H^{\mathsf{NM},(T)}(x,x)$, because by (4) there exists a vertex $z \in \mathcal{V}_H$ such that $\chi_G^{\mathsf{NM},(T)}(u,u) = \chi_H^{\mathsf{NM},(T)}(x,z)$, implying $0 = \mathrm{dis}_G(u,u) = \mathrm{dis}_H(x,z)$ by Lemma E.4.
- If $\mathrm{dis}_G(u,v) = \mathrm{dis}_H(x,y) = 0$, then $\mathcal{N}_{G^u}(v) = \emptyset$ and $\mathcal{N}_{H^x}(y) = \emptyset$.

In all cases (5) holds, which concludes the induction step. We finally obtain that the stable mappings satisfy $\chi^{\mathsf{NM}} \preceq \chi^{\mathsf{NDM}}$ and thus NM is more powerful than NDM (by Remark E.2(b)).

We next turn to the case $\mathsf{EGO}(k)$+NM vs. $\mathsf{EGO}(k)$. This case follows by a simple induction that $\chi^{\mathsf{EGO}(k)+\mathsf{NM},(t)} \preceq \chi^{\mathsf{EGO}(k),(t)}$ for all $t \in \mathbb{N}$.

We finally turn to the case NDM vs. ND. This case also follows by a simple induction that $\chi^{\mathsf{NDM},(t)} \preceq \chi^{\mathsf{ND},(t)}$ for all $t \in \mathbb{N}$. □

It remains to prove the following key lemma:

Lemma E.4. *Consider an SWL algorithm* A *such that the aggregation scheme* $\mathcal{A}$ *contains the two basic aggregations* $\mathsf{agg}_\mathsf{uv}^\mathsf{P}$ *and* $\mathsf{agg}_\mathsf{u}^\mathsf{L}$ *in Definition 3.1, and the node marking policy is used (possibly along with an ego network policy). Denote* $\chi^{(t)}$ *as the color mapping of* A *at iteration* t*. For any graphs* $G = (\mathcal{V}_G, \mathcal{E}_G)$*,* $H = (\mathcal{V}_H, \mathcal{E}_H)$ *and vertices* $u, v \in \mathcal{V}_G$*,* $x, y \in \mathcal{V}_H$*, when* $t \geq \min(D_G, D_H)$*,* $\chi_G^{(t)}(u,v) = \chi_H^{(t)}(x,y) \implies \mathrm{dis}_{G^u}(u,v) = \mathrm{dis}_{G^x}(x,y)$*. Here,* D_G *and* D_H *are the diameter of graphs* G *and* H*, respectively.*

Proof. It suffices to prove the following result: if $\mathrm{dis}_{G^u}(u,v) \neq \mathrm{dis}_{G^x}(x,y)$, then $\chi_G^{(t)}(u,v) \neq \chi_H^{(t)}(x,y)$ for $t = \min(\mathrm{dis}_{G^u}(u,v), \mathrm{dis}_{G^x}(x,y))$. We prove it by induction over t.

For the base case of $t = 0$, without loss of generality we can assume $\mathrm{dis}_{G^u}(u,v) = 0$ and $\mathrm{dis}_{H^x}(x,y) > 0$. For node marking policy, we clearly have $\chi_G^{(0)}(u,v) \neq \chi_H^{(0)}(x,y)$ since $v = u$ but $y \neq x$.

Assume the result holds for $t \leq T$ and consider $t = T+1$. Without loss of generality, we can assume $\mathrm{dis}_{G^u}(u,v) = T+1$ and $\mathrm{dis}_{H^x}(x,y) > T+1$ (remark: $\mathrm{dis}_{H^x}(x,y)$ can be ∞ for ego network policy). If $\chi_G^{(T+1)}(u,v) = \chi_H^{(T+1)}(x,y)$, by definition of $\mathsf{agg}_\mathsf{u}^\mathsf{L}$ we have

$$\{\!\{\chi_G^{(T)}(u,w) : w \in \mathcal{N}_{G^u}(v)\}\!\} = \{\!\{\chi_H^{(T)}(x,z) : z \in \mathcal{N}_{H^x}(y)\}\!\}.$$

Pick any vertex $w \in \mathcal{N}_{G^u}(v)$ satisfying $\mathrm{dis}_{G^u}(u,w)+1=\mathrm{dis}_{G^u}(u,v)$. Then, there is a vertex $z \in \mathcal{N}_{H^x}(y)$ such that $\chi_G^{(T)}(u,w)=\chi_H^{(T)}(x,z)$. By induction, $\mathrm{dis}_{G^u}(u,w)=\mathrm{dis}_{H^x}(x,z)$. This yields a contradiction, since

$$\mathrm{dis}_{H^x}(x,y) \le \mathrm{dis}_{H^x}(x,z)+1=\mathrm{dis}_{G^u}(u,w)+1=\mathrm{dis}_{G^u}(u,v)=T+1.$$

This concludes the induction step. □

The above lemma directly leads to the following corollary, which is useful in subsequent analysis.

Corollary E.5. *Let χ be the stable color mapping of SWL algorithm $\mathsf{A}(\mathcal{A},\mathsf{Pool})$ defined in Definition 4.3, satisfying $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$. For any graphs $G=(\mathcal{V}_G,\mathcal{E}_G)$, $H=(\mathcal{V}_H,\mathcal{E}_H)$ and vertices $u,v \in \mathcal{V}_G$, $x,y \in \mathcal{V}_H$, if $\chi_G(u,v)=\chi_H(x,y)$, then*

- $\mathrm{dis}_G(u,v)=\mathrm{dis}_H(x,y)$;
- $\chi_G(u,u)=\chi_H(x,x)$.

Proof. The first bullet directly follows from Lemma E.4. The second bullet can be proved by induction over the distance $\mathrm{dis}_G(u,v)$. For the base case of $\mathrm{dis}_G(u,v)=\mathrm{dis}_H(x,y)=0$, $u=v$, $x=y$, and the result already holds. For the induction step, the proof is similar to the above proof of Lemma E.4, in that we can find $w \in \mathcal{N}_G(v)$ and $z \in \mathcal{N}_H(y)$ such that $\mathrm{dis}_G(u,w)+1=\mathrm{dis}_G(u,v)$ and $\mathrm{dis}_H(x,z)+1=\mathrm{dis}_H(x,y)$, and $\chi_G(u,w)=\chi_H(x,z)$ (we omit the detail here). This finishes the induction step and concludes the proof. □

E.3. Hierarchy of different aggregation schemes

This subsection gives a complete analysis of different aggregation schemes in SWL, which is related to the proofs of Theorem 4.4. Note that we focus on the canonical node marking policy with a fixed pooling paradigm Pool throughout this subsection. In all proofs, we denote $G=(\mathcal{V}_G,\mathcal{E}_G)$ and $H=(\mathcal{V}_H,\mathcal{E}_H)$ as any connected graphs.

Lemma E.6. *Let χ be the stable color mapping of SWL algorithm $\mathsf{A}(\mathcal{A},\mathsf{Pool})$ defined in Definition 4.3, satisfying $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$. For any vertices $u \in \mathcal{V}_G$ and $x \in \mathcal{V}_H$, if $\chi_G(u,u)=\chi_H(x,x)$, then $\{\!\{\chi_G(u,v): v \in \mathcal{V}_G\}\!\}=\{\!\{\chi_H(x,y): y \in \mathcal{V}_H\}\!\}$.*

Proof. Actually, we can prove a stronger result: for any $k \in \mathbb{N}$,

$$\{\!\{\chi_G(u,v): v \in \mathcal{N}_G^k(u)\}\!\}=\{\!\{\chi_H(x,y): y \in \mathcal{N}_H^k(x)\}\!\}. \tag{6}$$

This implies Lemma E.6 because G and H are connected graphs.

We prove it by induction over k. The base case of $k=0$ is trivial. Now assume (6) holds for all $k \le K$, and we want to prove that (6) holds for $k=K+1$. Using the condition $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$, for any vertices $v \in \mathcal{V}_G$ and $y \in \mathcal{V}_H$ satisfying $\chi_G(u,v)=\chi_H(x,y)$, we have

$$\{\!\{\chi_G(u,w): w \in \mathcal{N}_G(v)\}\!\}=\{\!\{\chi_H(x,z): z \in \mathcal{N}_H(y)\}\!\}. \tag{7}$$

Combining (7) with (6), we obtain

$$\bigcup_{v \in \mathcal{D}_G^K(u)} \{\!\{\chi_G(u,w): w \in \mathcal{N}_G(v)\}\!\}=\bigcup_{y \in \mathcal{D}_H^K(x)} \{\!\{\chi_H(x,z): z \in \mathcal{N}_H(y)\}\!\}, \tag{8}$$

where we define

$$\mathcal{D}_G^K(u):=\mathcal{N}_G^K(u)\backslash\mathcal{N}_G^{K-1}(u)=\{v \in \mathcal{V}_G: \mathrm{dis}_G(u,v)=K\}.$$

Here, each vertex w in (8) satisfies $K-1 \le \mathrm{dis}_G(u,w) \le K+1$, and each vertex z in (8) satisfies $K-1 \le \mathrm{dis}_H(x,z) \le K+1$. By Corollary E.5, for any vertices $w \in \mathcal{V}_G$ and $z \in \mathcal{V}_H$, $\mathrm{dis}_G(u,w) \ne \mathrm{dis}_H(x,z)$ implies $\chi_G(u,w) \ne \chi_H(x,z)$. Therefore,

$$\bigcup_{v \in \mathcal{D}_G^K(u)} \{\!\{\chi_G(u,w): w \in \mathcal{N}_G(v) \cap \mathcal{D}_G^{K+1}(u)\}\!\}=\bigcup_{y \in \mathcal{D}_H^K(x)} \{\!\{\chi_H(x,z): z \in \mathcal{N}_H(y) \cap \mathcal{D}_H^{K+1}(x)\}\!\}. \tag{9}$$

Rearranging the terms in (9) yields an equivalent formula:

$$\bigcup_{w\in\mathcal{D}_G^{K+1}(u)} \{\!\{\chi_G(u,w)\}\!\} \times |\mathcal{N}_G(w)\cap\mathcal{D}_G^K(u)| = \bigcup_{z\in\mathcal{D}_H^{K+1}(x)} \{\!\{\chi_H(x,z)\}\!\} \times |\mathcal{N}_H(z)\cap\mathcal{D}_H^K(x)|, \tag{10}$$

where we denote $\{\!\{c\}\!\} \times M$ as a multiset containing M repeated elements c. Next, note that if $\chi_G(u,w) = \chi_H(x,z)$ for some $w \in \mathcal{V}_G$ and $z \in \mathcal{V}_H$, then by (7) and Corollary E.5, we have $|\mathcal{N}_G(w)\cap\mathcal{D}_G^K(u)| = |\mathcal{N}_H(z)\cap\mathcal{D}_H^K(x)|$. This proves

$$\{\!\{\chi_G(u,w) : w \in \mathcal{N}_G^{K+1}(u)\}\!\} = \{\!\{\chi_H(x,z) : z \in \mathcal{N}_H^{K+1}(x)\}\!\}$$

and finishes the induction step. □

Corollary E.7. *Let χ^{L}, χ^{G}, and χ^{LG} be the stable color mappings of SWL algorithms $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}\},\mathsf{Pool})$, $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\},\mathsf{Pool})$ and $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}},\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\},\mathsf{Pool})$, respectively. Then, $\chi^{\mathsf{LG}} \simeq \chi^{\mathsf{L}} \preceq \chi^{\mathsf{G}}$.*

Proof. The proof is based on Remark E.2(c). We first prove that $\chi^{\mathsf{L}} \preceq \chi^{\mathsf{G}}$. Define an auxiliary color mapping $\tilde{\chi} = \mathscr{T}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\},\chi^{\mathsf{L}})$ where $\mathscr{T}$ is defined in (3). It suffices to prove that $\chi^{\mathsf{L}} \preceq \tilde{\chi}$.

Consider any vertices $u,v \in \mathcal{V}_G$ and $x,y \in \mathcal{V}_H$ satisfying $\chi_G^{\mathsf{L}}(u,v) = \chi_H^{\mathsf{L}}(x,y)$. Since the mapping χ^{L} is already stable, for any $\mathsf{agg} \in \mathcal{A}$, we have

$$\mathsf{agg}(u,v,G,\chi_G^{\mathsf{L}}) = \mathsf{agg}(x,y,H,\chi_H^{\mathsf{L}}).$$

Moreover, due to the use of $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}$, by Corollary E.5 we have $\chi_G^{\mathsf{L}}(u,u) = \chi_H^{\mathsf{L}}(x,x)$. Using Lemma E.6, we further obtain

$$\{\!\{\chi_G^{\mathsf{L}}(u,w) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H^{\mathsf{L}}(x,z) : z \in \mathcal{V}_H\}\!\}.$$

Namely,

$$\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}(u,v,G,\chi_G^{\mathsf{L}}) = \mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}(x,y,H,\chi_H^{\mathsf{L}}).$$

Therefore, $\tilde{\chi}(u,v) = \tilde{\chi}(x,y)$. We have proved $\chi^{\mathsf{L}} \preceq \tilde{\chi}$.

We next turn to $\chi^{\mathsf{L}} \simeq \chi^{\mathsf{LG}}$, for which it suffices to prove $\chi^{\mathsf{L}} \preceq \chi^{\mathsf{LG}}$. The process is exactly the same as above. □

Lemma E.8. *Let χ be the stable color mapping of SWL algorithm $\mathsf{A}(\mathcal{A},\mathsf{Pool})$ defined in Definition 4.3, satisfying $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}} \in \mathcal{A}$. For any vertices $u,v \in \mathcal{V}_G$ and $x,y \in \mathcal{V}_H$, if $\chi_G(u,v) = \chi_H(x,y)$, then $\chi_G(u,u) = \chi_H(x,x)$.*

Proof. Since $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}} \in \mathcal{A}$, we have

$$\{\!\{\chi_G(u,w) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(x,z) : z \in \mathcal{V}_H\}\!\}.$$

Therefore, there is a vertex $z \in \mathcal{V}_H$ such that $\chi_G(u,u) = \chi_H(x,z)$. By definition of node marking policy, we must have $x = z$ (otherwise, the initial color satisfies $\chi_G^{(0)}(u,u) \neq \chi_H^{(0)}(x,z)$, a contradiction). This already proves that $\chi_G(u,u) = \chi_H(x,x)$. □

Corollary E.9. *Let χ^{G}, χ^{P}, and χ^{GP} be the stable color mappings of SWL algorithms $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\},\mathsf{Pool})$, $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}\},\mathsf{Pool})$, and $\mathsf{A}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}},\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}\},\mathsf{Pool})$, respectively. Then, $\chi^{\mathsf{GP}} \simeq \chi^{\mathsf{G}} \preceq \chi^{\mathsf{P}}$.*

Proof. Similar to Corollary E.7, the proof is based on Remark E.2(c). We only prove $\chi^{\mathsf{G}} \preceq \chi^{\mathsf{P}}$, and the proof of $\chi^{\mathsf{G}} \preceq \chi^{\mathsf{GP}}$ is exactly the same. Define an auxiliary color mapping $\tilde{\chi} = \mathscr{T}(\mathcal{A}\cup\{\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}\},\chi^{\mathsf{G}})$ where $\mathscr{T}$ is defined in (3). It suffices to prove that $\chi^{\mathsf{G}} \preceq \tilde{\chi}$.

Consider any vertices $u,v \in \mathcal{V}_G$ and $x,y \in \mathcal{V}_H$ satisfying $\chi_G^{\mathsf{G}}(u,v) = \chi_H^{\mathsf{G}}(x,y)$. Due to the presence of $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$, by Lemma E.8 we have $\chi_G^{\mathsf{G}}(u,u) = \chi_H^{\mathsf{G}}(x,x)$. This already implies

$$\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}(u,v,G,\chi_G^{\mathsf{G}}) = \mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}(x,y,H,\chi_H^{\mathsf{G}}).$$

For any $\mathsf{agg} \in \mathcal{A}$, we also have

$$\mathsf{agg}(u,v,G,\chi_G^{\mathsf{G}}) = \mathsf{agg}(x,y,H,\chi_H^{\mathsf{G}}).$$

Therefore, $\tilde{\chi}(u,v) = \tilde{\chi}(x,y)$, namely, $\chi^{\mathsf{G}} \preceq \tilde{\chi}$. □

Lemma E.10. *Let χ be the stable color mapping of SWL algorithm $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool})$. Then for any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$, if $\chi_G(u, v) = \chi_H(x, y)$, then $\chi_G(v, u) = \chi_H(y, x)$.*

Proof. We will prove a stronger result: let $\chi^{(t)}$ be the color mapping at iteration t, then for any $t \in \mathbb{N}$, $\chi^{(t)}_G(u, v) = \chi^{(t)}_H(x, y) \iff \chi^{(t)}_G(v, u) = \chi^{(t)}_H(y, x)$. We prove it by induction over t.

The base case of $t = 0$ trivially holds by definition of the node marking. Assume the above result holds for $t = T$, and consider $t = T + 1$. Let $\chi^{(T+1)}_G(u, v) = \chi^{(T+1)}_H(x, y)$. By definition of the aggregation scheme $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}$, we have

$$\{\!\{\chi^{(T)}_G(u, w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi^{(T)}_H(x, z) : z \in \mathcal{N}_H(y)\}\!\},$$
$$\{\!\{\chi^{(T)}_G(w, v) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{\chi^{(T)}_H(z, y) : z \in \mathcal{N}_H(x)\}\!\}.$$

Using induction we obtain

$$\{\!\{\chi^{(T)}_G(w, u) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi^{(T)}_H(z, x) : z \in \mathcal{N}_H(y)\}\!\},$$
$$\{\!\{\chi^{(T)}_G(v, w) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{\chi^{(T)}_H(y, z) : z \in \mathcal{N}_H(x)\}\!\}.$$

Therefore, $\chi^{(T+1)}_G(v, u) = \chi^{(T+1)}_H(y, x)$, which finishes the induction step. □

Corollary E.11. *Let χ^{LL}, χ^{LP}, and χ^{LLP} be the stable color mappings of SWL algorithms $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool})$, $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool})$, and $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool})$, respectively. Then, $\chi^{\mathsf{LL}} \simeq \chi^{\mathsf{LP}} \simeq \chi^{\mathsf{LLP}}$.*

Proof. Similar to Corollary E.7, the proof is based on Remark E.2(c). We only prove $\chi^{\mathsf{LL}} \simeq \chi^{\mathsf{LP}}$, and the proof of $\chi^{\mathsf{LL}} \simeq \chi^{\mathsf{LLP}}$ is exactly the same.

We first prove $\chi^{\mathsf{LP}} \preceq \chi^{\mathsf{LL}}$. Define an auxiliary color mapping $\tilde{\chi} = \mathscr{T}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \chi^{\mathsf{LP}})$ where $\mathscr{T}$ is defined in (3). It suffices to prove that $\chi^{\mathsf{LP}} \preceq \tilde{\chi}$. Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi^{\mathsf{LP}}_G(u, v) = \chi^{\mathsf{LP}}_H(x, y)$. Since the mapping χ^{LP} is already stable, we have $\chi^{\mathsf{LP}}_G(v, u) = \chi^{\mathsf{LP}}_H(y, x)$ and

$$\{\!\{\chi^{\mathsf{LP}}_G(u, w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi^{\mathsf{LP}}_H(x, z) : z \in \mathcal{N}_H(y)\}\!\}. \tag{11}$$

Since $\chi^{\mathsf{LP}}_G(v, u) = \chi^{\mathsf{LP}}_H(y, x)$, we also have

$$\{\!\{\chi^{\mathsf{LP}}_G(v, w) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{\chi^{\mathsf{LP}}_H(y, z) : z \in \mathcal{N}_H(x)\}\!\}.$$

This further implies

$$\{\!\{\chi^{\mathsf{LP}}_G(w, v) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{\chi^{\mathsf{LP}}_H(z, y) : z \in \mathcal{N}_H(x)\}\!\}. \tag{12}$$

Combining (11) and (12) we obtain $\tilde{\chi}_G(u, v) = \tilde{\chi}_H(x, y)$, as desired.

We next prove $\chi^{\mathsf{LL}} \preceq \chi^{\mathsf{LP}}$. Define an auxiliary color mapping $\tilde{\chi} = \mathscr{T}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \chi^{\mathsf{LL}})$ where $\mathscr{T}$ is defined in (3). It suffices to prove that $\chi^{\mathsf{LL}} \preceq \tilde{\chi}$. This simply follows by the fact that the stable color mapping χ^{LL} cannot be refined using $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ (by definition) or using $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$ (by Lemma E.10). □

E.4. Analyzing the pooling paradigm

This subsection discusses how the pooling paradigm can influence the expressive power of the SWL algorithm, which is related to the proofs of Theorem 4.6. In all proofs, we denote $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ as any graphs.

Lemma E.12. *Let $\mathcal{A}$ be defined in Definition 4.3 with $\{\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\} \cap \mathcal{A} \neq \emptyset$. Then, $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{SV})$.*

Proof. Let $c^{\mathsf{VS}}(G)$ and $c^{\mathsf{SV}}(G)$ be the graph representations computed by algorithms $\mathsf{A}(\mathcal{A}, \mathsf{VS})$ and $\mathsf{A}(\mathcal{A}, \mathsf{SV})$, respectively. Since both algorithms use the same aggregation scheme, we denote the stable color mapping as χ. We aim to prove that if $c^{\mathsf{SV}}(G) = c^{\mathsf{SV}}(H)$, then $c^{\mathsf{VS}}(G) = c^{\mathsf{VS}}(H)$.

Let $c^{\mathsf{SV}}(G) = c^{\mathsf{SV}}(H)$, then by definition of SV pooling

$$\{\!\{r^{\mathsf{V}}_G(v) : v \in \mathcal{V}_G\}\!\} = \{\!\{r^{\mathsf{V}}_H(y) : y \in \mathcal{V}_H\}\!\},$$

where we denote $r_G^{\mathsf{V}}(v) = \{\!\{\chi_G(u,v) : u \in \mathcal{V}_G\}\!\}$. Consider any vertices $v \in \mathcal{V}_G$ and $y \in \mathcal{V}_H$ satisfying $r_G^{\mathsf{V}}(v) = r_H^{\mathsf{V}}(y)$. Then, there exists vertex $w \in \mathcal{V}_H$ such that $\chi_G(v,v) = \chi_H(w,y)$. Due to the definition of node marking, we must have $w = y$. This implies that $\chi_G(v,v) = \chi_H(y,y)$. Now separately consider two cases:

- If $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}} \in \mathcal{A}$, then by definition of stable color mapping we have $\{\!\{\chi_G(v,u) : u \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(y,x) : x \in \mathcal{V}_H\}\!\}$;
- If $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$, then by Lemma E.6 we have $\{\!\{\chi_G(v,u) : u \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(y,x) : x \in \mathcal{V}_H\}\!\}$.

In both cases, we have $r_G^{\mathsf{S}}(v) = r_H^{\mathsf{S}}(y)$ where we denote $r_G^{\mathsf{S}}(v) = \{\!\{\chi_G(v,u) : u \in \mathcal{V}_G\}\!\}$.

Therefore, we have proved that $r_G^{\mathsf{V}}(v) = r_H^{\mathsf{V}}(y) \implies r_G^{\mathsf{S}}(v) = r_H^{\mathsf{S}}(y)$. This finally yields

$$\{\!\{r_G^{\mathsf{S}}(v) : v \in \mathcal{V}_G\}\!\} = \{\!\{r_H^{\mathsf{S}}(y) : y \in \mathcal{V}_H\}\!\},$$

namely, $c^{\mathsf{VS}}(G) = c^{\mathsf{VS}}(H)$, as desired. □

E.5. Proof of theorems in Section 4.2

We are now ready to prove all the main results in Section 4.2, which we restate below.

Theorem 4.4. *Under the notation of Definition 4.3, the following hold:*

- $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}\}, \mathsf{Pool})$ *and* $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{Pool})$*;*
- $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{Pool})$ *and* $\mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\mathcal{A} \cup \{\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}, \mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}\}\}, \mathsf{Pool})$*;*
- $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vu}}^{\mathsf{P}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vu}}^{\mathsf{P}}\}, \mathsf{Pool})$.

Proof. Based on Remark E.2(b), we only need to focus on the stable color mappings of these algorithms. The proof readily follows by using Corollaries E.7, E.9 and E.11. □

Proposition 4.5. *Let $\mathcal{A}$ be any aggregation scheme defined in Definition 4.3. Denote $\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}$ as the aggregation scheme obtained from $\mathcal{A}$ by exchanging the element $\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}$ with $\mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}$, exchanging $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}$ with $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$, and exchanging $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$ with $\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}$. Then, $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \simeq \mathsf{A}(\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}, \mathsf{SV})$.*

Proof. The proof is almost trivial by symmetry. It is easy to see that for any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$, $\chi_G(u,v) = \chi_H(x,y) \iff \chi_G^{\mathsf{u}\leftrightarrow\mathsf{v}}(v,u) = \chi_H^{\mathsf{u}\leftrightarrow\mathsf{v}}(y,x)$, where χ and $\chi^{\mathsf{u}\leftrightarrow\mathsf{v}}$ are the stable color mapping of SWL algorithms $\mathsf{A}(\mathcal{A}, \mathsf{VS})$ and $\mathsf{A}(\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}, \mathsf{SV})$, respectively. □

Theorem 4.6. *Let $\mathcal{A}$ be defined in Definition 4.3 with $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$. Then the following hold:*

- $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{SV})$*;*
- *If* $\{\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\} \cap \mathcal{A} \neq \emptyset$*, then* $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \simeq \mathsf{A}(\mathcal{A}, \mathsf{SV})$.

Proof. The first bullet is a direct consequence of Lemma E.12. The second bullet is a direct consequence of the first bullet and Proposition 4.5, since we have both $\mathsf{A}(\mathcal{A}, \mathsf{VS}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{SV})$ and $\mathsf{A}(\mathcal{A}, \mathsf{SV}) \simeq \mathsf{A}(\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}, \mathsf{VS}) \preceq \mathsf{A}(\mathcal{A}^{\mathsf{u}\leftrightarrow\mathsf{v}}, \mathsf{SV}) \simeq \mathsf{A}(\mathcal{A}, \mathsf{VS})$. □

Corollary 4.7. *Let $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ be any SWL algorithm defined in Definition 4.3 with at least one local aggregation, i.e. $\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\} \cap \mathcal{A} \neq \emptyset$. Then, $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ must be as expressive as one of the 6 SWL algorithms defined below:*

- (Vanilla SWL) $\mathsf{SWL}(\mathsf{VS}) := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}\}, \mathsf{VS})$, $\mathsf{SWL}(\mathsf{SV}) := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}\}, \mathsf{SV})$;
- (SWL with additional single-point aggregation) $\mathsf{PSWL}(\mathsf{VS}) := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}\}, \mathsf{VS})$, $\mathsf{PSWL}(\mathsf{SV}) := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}\}, \mathsf{SV})$;
- (SWL with additional global aggregation) $\mathsf{GSWL} := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}\}, \mathsf{VS})$;
- (Symmetrized SWL) $\mathsf{SSWL} := \mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\}, \mathsf{VS})$.

Moreover, we have

$$\mathsf{SWL}(\mathsf{VS}) \preceq \mathsf{SWL}(\mathsf{SV}) \textit{ and } \mathsf{PSWL}(\mathsf{VS}) \preceq \mathsf{PSWL}(\mathsf{SV}),$$
$$\mathsf{SWL}(\mathsf{VS}) \preceq \mathsf{PSWL}(\mathsf{VS}) \textit{ and } \mathsf{SWL}(\mathsf{SV}) \preceq \mathsf{PSWL}(\mathsf{SV}),$$
$$\mathsf{PSWL}(\mathsf{SV}) \preceq \mathsf{GSWL} \preceq \mathsf{SSWL}.$$

Proof. Due to Proposition 4.5, we can assume $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}} \in \mathcal{A}$ without loss of generality. We separately consider several cases:

- Case 1: $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\} \cap \mathcal{A} = \emptyset$. In this case, we have

$$\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \preceq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{Pool})$$

  by Theorem 4.4. On the other hand, clearly $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{Pool})$. We thus have $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}\}, \mathsf{Pool})$, namely, $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{SWL}(\mathsf{VS})$ or $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{SWL}(\mathsf{SV})$.
- Case 2: $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}} \in \mathcal{A}$ and $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\} \cap \mathcal{A} = \emptyset$. In this case, we have

$$\begin{aligned}\mathsf{A}(\mathcal{A}, \mathsf{Pool}) &\preceq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool})\end{aligned}$$

  by Theorem 4.4. On the other hand, clearly $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{Pool})$. We thus have $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool})$, namely, $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{PSWL}(\mathsf{VS})$ or $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{PSWL}(\mathsf{SV})$.
- Case 3: $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}} \in \mathcal{A}$ and $\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\} \cap \mathcal{A} = \emptyset$. In this case, we have

$$\begin{aligned}\mathsf{A}(\mathcal{A}, \mathsf{Pool}) &\preceq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{Pool})\end{aligned}$$

  by Theorem 4.4. On the other hand, clearly $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{Pool})$. We thus have $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{Pool})$. Moreover, by Theorem 4.6 we have $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{VS}) = \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}\}, \mathsf{SV})$. Therefore, $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{GSWL}$.
- Case 4: $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}} \in \mathcal{A}$ or $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}} \in \mathcal{A}$. In this cases, a similar analysis yields

$$\begin{aligned}\mathsf{A}(\mathcal{A}, \mathsf{Pool}) &\preceq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool}) \\ &\simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool})\end{aligned}$$

  by Theorem 4.4. On the other hand, $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{Pool}) \preceq \mathsf{A}(\mathcal{A}, \mathsf{Pool})$. We thus have $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{Pool})$. Moreover, by Theorem 4.6 we have $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{VS}) = \mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \mathsf{SV})$. Therefore, $\mathsf{A}(\mathcal{A}, \mathsf{Pool}) \simeq \mathsf{SSWL}$.

Combining the four cases concludes the proof. □

F. Discussions on Subgraph GNNs beyond the Framework of Definition 2.1

There have been several prior works that design subgraph GNNs beyond the aggregation schemes of Definition 2.1. In this section, we will investigate them and compare the expressive power with our framework. We focus on the WL algorithm corresponding to each subgraph GNN because it has the same expressive power as the GNN model in distinguishing non-isomorphic graphs (which can be easily proved following Appendix D). Throughout this section, we assume that node marking policy is used since it achieves the strongest expressive power according to Proposition 4.2.

Below, we discuss the following works:

- GNN-AK (Zhao et al., 2022a);
- GNN-AK-ctx (Zhao et al., 2022a);
- DSS-WL (Bevilacqua et al., 2022);
- SUN (Frasca et al., 2022);
- RelGN(2) (Frasca et al., 2022).

GNN-AK (Zhao et al., 2022a). The GNN aggregation scheme can be written as

$$\chi_G^{(t+1)}(u,v) = \begin{cases} \mathsf{hash}(\chi_G^{(t)}(u,v), \\ \qquad \chi_G^{(t)}(v,v), & \text{if } u \neq v, \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\}) \\ \mathsf{hash}(\chi_G^{(t)}(v,v), \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\}, & \text{if } u = v. \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{V}_G\}\!\}) \end{cases}$$

GNN-AK uses the vertex-subgraph pooling. As can be seen, there is an additional global aggregation $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$ when $u = v$, which differs from the case of $u \neq v$. Therefore, it goes beyond the framework of Definition 3.1.

Proposition F.1. GNN-AK *is as powerful as* PSWL(VS).

Proof. Consider the following two SWL algorithms defined in Definition 4.3: (i) $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}\}, \mathsf{VS})$, and (ii) $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}, \mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{VS})$. It is clear that the stable color mapping of GNN-AK is finer than that of $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}\}, \mathsf{VS})$, but the stable color mapping of $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}, \mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}\}, \mathsf{VS})$ is finer than GNN-AK. However, both algorithms are equivalent to PSWL(VS) as shown in Corollary 4.7. Therefore, by Remark E.2(b) GNN-AK is as powerful as PSWL(VS). □

GNN-AK-ctx (Zhao et al., 2022a). The GNN aggregation scheme can be written as

$$\chi_G^{(t+1)}(u,v) = \begin{cases} \mathsf{hash}(\chi_G^{(t)}(u,v), \\ \qquad \chi_G^{(t)}(v,v), & \text{if } u \neq v, \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\}) \\ \mathsf{hash}(\chi_G^{(t)}(v,v), \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\}, \\ \qquad \{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{V}_G\}\!\}, & \text{if } u = v. \\ \qquad \{\!\{\chi_G^{(t)}(w,v) : w \in \mathcal{V}_G\}\!\}) \end{cases}$$

GNN-AK also uses the vertex-subgraph pooling. Compared with GNN-AK, GNN-AK-ctx further introduces the cross-graph global aggregation $\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}$ when $u = v$ (which they called the *contextual encoding*).

Proposition F.2. GNN-AK-ctx *is as powerful as* GSWL.

Proof. Similar to the above proof, by using the result that GSWL is as powerful as $\mathsf{A}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}\}, \mathsf{VS})$ (Corollary 4.7), it is clear that GSWL is more powerful than GNN-AK-ctx. It remains to prove that GNN-AK-ctx is more powerful than GSWL.

The proof is based on Remark E.2(c). Let χ be the stable color mapping of GNN-AK-ctx. Define an auxiliary color mapping $\tilde{\chi} = \mathscr{T}(\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}\}, \chi)$ where $\mathscr{T}$ is defined in (3). It suffices to prove that $\chi \preceq \tilde{\chi}$.

Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi_G(u,v) = \chi_H(x,y)$. Since the mapping χ is already stable, by definition of GNN-AK-ctx we have

$$\chi_G(v,v) = \chi_H(y,y), \tag{13}$$

$$\{\!\{\chi_G(u,w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi_H(x,z) : z \in \mathcal{N}_H(y)\}\!\}. \tag{14}$$

Again by definition of the stable color mapping, (13) implies that

$$\{\!\{\chi_G(w,v) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(z,y) : z \in \mathcal{V}_H\}\!\}. \tag{15}$$

Combining with (14) and (15), we obtain that $\tilde{\chi}_G(u,v) = \tilde{\chi}_H(x,y)$, concluding the proof. □

DSS-WL (Bevilacqua et al., 2022). The aggregation scheme of DSS-WL can be written as

$$\begin{aligned}\chi_G^{(t+1)}(u,v) = \mathsf{hash}(&\chi_G^{(t)}(u,v),\\ &\{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\},\\ &\{\!\{\chi_G^{(t)}(w,v) : w \in \mathcal{V}_G\}\!\},\\ &\{\!\{\chi_G^{(t)}(w,w') : w \in \mathcal{V}_G, w' \in \mathcal{N}_G(v)\}\!\}).\end{aligned}$$

Here, the last aggregation does not belong to Definition 3.1. DSS-WL also uses the vertex-subgraph pooling.

Proposition F.3. DSS-WL *is as powerful as* GSWL.

Proof. Clearly, DSS-WL is more powerful than the SWL algorithm $\mathsf{A}(\{\mathsf{agg}_\mathsf{u}^\mathsf{L}, \mathsf{agg}_\mathsf{v}^\mathsf{G}\}, \mathsf{VS})$, which is precisely GSWL. It remains to prove that GSWL is more powerful than DSS-WL.

Similar to Proposition F.2, the proof is based on Remark E.2(c). Let χ be the stable color mapping of GSWL. For any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$, if $\chi_G(u,v) = \chi_H(x,y)$, then we have

$$\{\!\{\chi_G(u,w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi_H(x,z) : z \in \mathcal{N}_H(y)\}\!\}, \tag{16}$$

$$\{\!\{\chi_G(w,v) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(z,y) : z \in \mathcal{V}_H\}\!\}. \tag{17}$$

Plugging (17) into (16) yields

$$\{\!\{\{\!\{\chi_G(w,w') : w \in \mathcal{V}_G\}\!\} : w' \in \mathcal{N}_G(v)\}\!\} = \{\!\{\{\!\{\chi_H(z,z') : z \in \mathcal{V}_H\}\!\} : z' \in \mathcal{N}_H(y)\}\!\}.$$

Therefore,

$$\{\!\{\chi_G(w,w') : w \in \mathcal{N}_G(v), w' \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(z,z') : w \in \mathcal{N}_H(y), z' \in \mathcal{V}_H\}\!\}. \tag{18}$$

Combining with (16), (17), and (18), it implies that DSS-WL cannot further refine the stable color mapping χ, which concludes the proof. □

SUN (Frasca et al., 2022). The WL aggregation scheme can be written as

$$\begin{aligned}\chi_G^{(t+1)}(u,v) = \mathsf{hash}(&\chi_G^{(t)}(u,v), \chi_G^{(t)}(u,u), \chi_G^{(t)}(v,v),\\ &\{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{N}_G(v)\}\!\},\\ &\{\!\{\chi_G^{(t)}(u,w) : w \in \mathcal{V}_G\}\!\},\\ &\{\!\{\chi_G^{(t)}(w,v) : w \in \mathcal{V}_G\}\!\},\\ &\{\!\{\chi_G^{(t)}(w,w') : w \in \mathcal{V}_G, w' \in \mathcal{N}_G(v)\}\!\}).\end{aligned}$$

We note that the formulation of Frasca et al. (2022) slightly differs from the above WL formula, in that SUN introduces different model parameters separately for the cases of $u = v$ and $u \neq v$, respectively. However, when using a node marking policy, introducing two sets of parameters for the two cases does not theoretically increase the expressivity (but it may benefit practical performance in real-world tasks).

Proposition F.4. SUN *is as powerful as* GSWL.

Proof. The proof is almost the same as the above one, by using the result that GSWL is as powerful as $\mathsf{A}(\{\mathsf{agg}_\mathsf{u}^\mathsf{L}, \mathsf{agg}_\mathsf{u}^\mathsf{G}, \mathsf{agg}_\mathsf{v}^\mathsf{G}, \mathsf{agg}_\mathsf{uu}^\mathsf{P}, \mathsf{agg}_\mathsf{vv}^\mathsf{P}\}, \mathsf{VS})$ (Corollary 4.7). We omit the details here. □

RelGN(2) (Frasca et al., 2022). This GNN architecture is motivated by 2-IGN (Maron et al., 2019b;c) by extending each basic equivariant linear operator into various types of local/global aggregations. Each atomic aggregation operation in RelGN(2) can be symbolized as $\mathsf{agg}^{\mathsf{op}_1,\mathsf{op}_2}$, where op_1 and op_2 can take one of the following symbols: Pu, Pv, G, Lu, Lv, and D. The semantic of $\mathsf{agg}^{\mathsf{op}_1,\mathsf{op}_2}$ is defined as follows:

$$\mathsf{agg}^{\mathsf{op}_1,\mathsf{op}_2}(u,v,G,\chi) = \{\!\{\chi(w,w') : w \in \bigcirc, w' \in \bigcirc\}\!\},$$

where the first/second $\bigcirc$ is filled by one of the following expression depending on op_1/op_2, respectively:

- For symbol Pu: $\bigcirc$ is filled by $\{u\}$;
- For symbol Pv: $\bigcirc$ is filled by $\{v\}$;
- For symbol Lu: $\bigcirc$ is filled by $\mathcal{N}_G(u)$;
- For symbol Lv: $\bigcirc$ is filled by $\mathcal{N}_G(v)$;
- For symbol G: $\bigcirc$ is filled by $\mathcal{V}_G$;
- For symbol D : $\bigcirc$ is filled by $\{w\}$. This symbol corresponds to diagonal aggregation and can only be used by op_2.

Based on the choice of op_1 and op_2, there are a total of $5 \times 6 - 2 = 28$ nonequivalent aggregation operations. Note that $\mathsf{agg}^{\mathsf{Pu},\mathsf{D}}$ is equivalent to $\mathsf{agg}^{\mathsf{Pu},\mathsf{Pu}}$ and $\mathsf{agg}^{\mathsf{Pv},\mathsf{D}}$ is equivalent to $\mathsf{agg}^{\mathsf{Pv},\mathsf{Pv}}$. As a result, $\mathsf{RelGN}(2)$ incorporates all these 28 aggregation operations into the WL iteration. Similar to SUN, $\mathsf{RelGN}(2)$ also introduces different model parameters separately for the cases of $u = v$ and $u \neq v$, respectively. It can be calculated that the total number of linear equivariant transformations is $28 + 11 = 39$.

Proposition F.5. $\mathsf{RelGN}(2)$ *is as powerful as* SSWL.

Proof. First, it is obvious that $\mathsf{RelGN}(2)$ is more powerful than SSWL. Therefore, it remains to prove that SSWL is more powerful than $\mathsf{RelGN}(2)$. Similar to the previous propositions, the proof is based on Remark E.2(c). Let χ be the stable color mapping of SSWL. Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi_G(u, v) = \chi_H(x, y)$. Since SSWL is as powerful as $\mathsf{A}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{x}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}\}, \mathsf{VS})$, we have

$$
\begin{aligned}
\chi_G(u,u) &= \chi_H(x,x),\\
\chi_G(v,v) &= \chi_H(y,y),\\
\chi_G(v,u) &= \chi_H(y,x),\\
\{\!\{\chi_G(u,w) : w \in \mathcal{V}_G\}\!\} &= \{\!\{\chi_H(x,z) : z \in \mathcal{V}_H\}\!\},\\
\{\!\{\chi_G(w,v) : w \in \mathcal{V}_G\}\!\} &= \{\!\{\chi_H(z,y) : z \in \mathcal{V}_H\}\!\},\\
\{\!\{\chi_G(u,w) : w \in \mathcal{N}_G(v)\}\!\} &= \{\!\{\chi_H(x,z) : z \in \mathcal{N}_H(y)\}\!\},\\
\{\!\{\chi_G(w,v) : w \in \mathcal{N}_G(u)\}\!\} &= \{\!\{\chi_H(z,y) : z \in \mathcal{N}_H(F)\}\!\}.
\end{aligned}
$$

Using a similar proof technique as previous propositions, we can show that χ cannot be refined by all $\mathsf{agg}^{\mathsf{op}_1,\mathsf{op}_2}$. We list one representative example below.

The diagonal aggregation $\mathsf{agg}^{\mathsf{G},\mathsf{D}}$. Combining the fourth equation and the second equation above, we obtain

$$\{\!\{\chi_G(w,w) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(z,z) : z \in \mathcal{V}_H\}\!\},$$

as desired. □

G. Proof of Theorems in Section 5

This section aims to prove Theorem 5.2. Throughout this section, we denote $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ as any graphs. Denote χ^{PSWL}, χ^{SSWL}, χ^{FWL}, χ^{LFWL}, and χ^{SLFWL} as the stable color mappings of $\mathsf{PSWL}(\mathsf{VS})$, SSWL, $\mathsf{FWL}(2)$, $\mathsf{LFWL}(2)$, and $\mathsf{SLFWL}(2)$, respectively.

We begin with the following simple fact, which holds by definition of the isomorphism type.

Fact G.1. *Let $\chi \in \{\chi^{\mathsf{FWL}}, \chi^{\mathsf{LFWL}}, \chi^{\mathsf{SLFWL}}\}$. For any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$, if $\chi_G(u, v) = \chi_H(x, y)$, then:*

- $u = v \iff x = y$;
- $\{u, v\} \in \mathcal{E}_G \iff \{x, y\} \in \mathcal{E}_H$.

Lemma G.2. *The following relations hold:*

- $\chi^{\mathsf{LFWL}} \preceq \chi^{\mathsf{PSWL}}$;
- $\chi^{\mathsf{SLFWL}} \preceq \chi^{\mathsf{SSWL}}$;
- $\chi^{\mathsf{SLFWL}} \preceq \chi^{\mathsf{LFWL}}$;

- $\chi^{\mathsf{FWL}} \preceq \chi^{\mathsf{SLFWL}}$.

Proof. Note that all the FWL-type algorithms considered in Lemma G.2 use isomorphism type as initial colors, which is finer than node marking in SWL algorithms. In this case, it is straightforward to see that Remark E.2(c) still applies. Namely, it suffices to prove that the stable color mapping of each stronger algorithm cannot get refined using the aggregation scheme of the weaker algorithm.

We first prove $\chi^{\mathsf{LFWL}} \preceq \mathscr{T}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}\}, \chi^{\mathsf{LFWL}}) := \tilde{\chi}$, where $\mathscr{T}$ is defined in (3). Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi^{\mathsf{LFWL}}_G(u,v) = \chi^{\mathsf{LFWL}}_H(x,y)$. Then by definition,

$$\{\!\{(\chi^{\mathsf{LFWL}}_G(u,w), \chi^{\mathsf{LFWL}}_G(w,v)) : w \in \mathcal{N}^1_G(v)\}\!\} = \{\!\{(\chi^{\mathsf{LFWL}}_H(x,z), \chi^{\mathsf{LFWL}}_H(z,y)) : z \in \mathcal{N}^1_H(y)\}\!\}.$$

It must be the case that

$$(\chi^{\mathsf{LFWL}}_G(u,v), \chi^{\mathsf{LFWL}}_G(v,v)) = (\chi^{\mathsf{LFWL}}_H(x,y), \chi^{\mathsf{LFWL}}_H(y,y)),$$
$$\{\!\{(\chi^{\mathsf{LFWL}}_G(u,w), \chi^{\mathsf{LFWL}}_G(w,v)) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{(\chi^{\mathsf{LFWL}}_H(x,z), \chi^{\mathsf{LFWL}}_H(z,y)) : z \in \mathcal{N}_H(y)\}\!\},$$

due to Fact G.1. Therefore,

$$\chi^{\mathsf{LFWL}}_G(v,v) = \chi^{\mathsf{LFWL}}_H(y,y)$$

and

$$\{\!\{\chi^{\mathsf{LFWL}}_G(u,w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi^{\mathsf{LFWL}}_H(x,z) : z \in \mathcal{N}_H(y)\}\!\}.$$

Namely, $\tilde{\chi}_G(u,v) = \tilde{\chi}_H(x,y)$. This proves $\chi^{\mathsf{LFWL}} \preceq \chi^{\mathsf{PSWL}}$.

We next prove $\chi^{\mathsf{SLFWL}} \preceq \mathscr{T}(\{\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}, \mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}\}, \chi^{\mathsf{SLFWL}}) := \tilde{\chi}$. Consider any vertices $u, v \in \mathcal{V}_G$ and $x, y \in \mathcal{V}_H$ satisfying $\chi^{\mathsf{SLFWL}}_G(u,v) = \chi^{\mathsf{SLFWL}}_H(x,y)$. Then by definition,

$$\begin{aligned}&\{\!\{(\chi^{\mathsf{SLFWL}}_G(u,w), \chi^{\mathsf{SLFWL}}_G(w,v)) : w \in \mathcal{N}^1_G(u) \cup \mathcal{N}^1_G(v)\}\!\}\\ =&\{\!\{(\chi^{\mathsf{SLFWL}}_H(x,z), \chi^{\mathsf{SLFWL}}_H(z,y)) : z \in \mathcal{N}^1_H(x) \cup \mathcal{N}^1_H(y)\}\!\}.\end{aligned}$$

Using Fact G.1 we have

$$\{\!\{(\chi^{\mathsf{SLFWL}}_G(u,w), \chi^{\mathsf{SLFWL}}_G(w,v)) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{(\chi^{\mathsf{SLFWL}}_H(x,z), \chi^{\mathsf{SLFWL}}_H(z,y)) : z \in \mathcal{N}_H(F)\}\!\},$$
$$\{\!\{(\chi^{\mathsf{SLFWL}}_G(u,w), \chi^{\mathsf{SLFWL}}_G(w,v)) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{(\chi^{\mathsf{SLFWL}}_H(x,z), \chi^{\mathsf{SLFWL}}_H(z,y)) : z \in \mathcal{N}_H(y)\}\!\}.$$

Therefore,

$$\{\!\{\chi^{\mathsf{SLFWL}}_G(u,w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi^{\mathsf{SLFWL}}_H(x,z) : z \in \mathcal{N}_H(y)\}\!\},$$
$$\{\!\{\chi^{\mathsf{SLFWL}}_G(w,v) : w \in \mathcal{N}_G(u)\}\!\} = \{\!\{\chi^{\mathsf{SLFWL}}_H(z,y) : z \in \mathcal{N}_H(F)\}\!\}.$$

Namely, $\tilde{\chi}_G(u,v) = \tilde{\chi}_H(x,y)$. This proves $\chi^{\mathsf{SLFWL}} \preceq \chi^{\mathsf{SSWL}}$.

The third and fourth bullets follow exactly the same procedure, so we omit the proof for clarity. □

Lemma G.3. *Let* $\chi \in \{\chi^{\mathsf{FWL}}, \chi^{\mathsf{LFWL}}, \chi^{\mathsf{SLFWL}}\}$. *If*

$$\{\!\{\chi_G(u,v) : u, v \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(x,y) : x, y \in \mathcal{V}_H\}\!\},$$

then

$$\{\!\{\{\!\{\chi_G(u,v) : v \in \mathcal{V}_G\}\!\} : u \in \mathcal{V}_G\}\!\} = \{\!\{\{\!\{\chi_H(x,y) : y \in \mathcal{V}_H\}\!\} : x \in \mathcal{V}_H\}\!\}.$$

Proof. Based on the assumption of Lemma G.3 and Fact G.1, we have

$$\{\!\{\chi_G(u,u) : u \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(x,x) : x \in \mathcal{V}_H\}\!\}.$$

Therefore, it suffices to prove that for any vertices $u \in \mathcal{V}_G$ and $x \in \mathcal{V}_H$, if $\chi_G(u,u) = \chi_H(x,x)$, then

$$\{\!\{\chi_G(u,v) : v \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(x,y) : y \in \mathcal{V}_H\}\!\}. \tag{19}$$

Based on Lemma G.2, we have $\chi \preceq \chi^{\mathsf{PSWL}}$. Note that χ^{PSWL} incorporates the aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$. Therefore, Lemma E.6 applies. We can follow the same proof technique of Lemma E.6 to obtain (19). □

We are now ready to prove Theorem 5.2, which we restate below:

Theorem 5.2. *The following relations hold:*

- $\mathsf{LFWL}(2) \preceq \mathsf{SLFWL}(2) \preceq \mathsf{FWL}(2)$*;*
- $\mathsf{PSWL}(\mathsf{VS}) \preceq \mathsf{LFWL}(2)$*;*
- $\mathsf{SSWL} \preceq \mathsf{SLFWL}(2)$.

Proof. The first bullet readily follows from Lemma G.2 and Remark E.2(b). For the other two bullets, although these algorithms have different pooling paradigms, we have proved that the pooling paradigm of FWL-type algorithms is as powerful as the pooling paradigm VS (Lemma G.3). Therefore, the results hold by Lemma G.2. □

H. Proof of Theorems in Section 6

This section proves the equivalence between SWL/FWL-type algorithms and pebbling games. For ease of presentation, we first define several notations.

Let $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ be two graphs, and let u, v be two types of pebbles. For each type of pebbles u, the placement information can be represented by a vertex pair (u_G, u_H) where $u_G \in \mathcal{V}_G$ and $u_H \in \mathcal{V}_H$ are the corresponding vertices that hold pebble u. Without abuse of notation, we also use the symbol u to represent the placement information of pebble u, i.e. $u = (u_G, u_H)$.

We next define a game modified from Section 6, called the L-round (u, v)-pebbling game.

Definition H.1. Given aggregation scheme $\mathcal{A}$ and an integer $L \in \mathbb{N}$, define the L-round (u, v)-pebbling game $\mathsf{G}^{\mathcal{A},L}(u; v)$ as follows. Initially, pebbles u and v are already placed on graphs G and H according to specified locations $u = (u_G, u_H)$, $v = (v_G, v_H)$. The game has L rounds. In each round, Spoiler and Duplicator can change the position of u and v according to the game rules of $\mathcal{A}$ defined in Section 6. Spoiler wins if after certain round $0 \le l \le L$, the isomorphism type of vertex pair (u_G, v_G) in graph G differs from the isomorphism type of vertex pair (u_H, v_H) in graph H. Duplicator wins the game if Spoiler does not win after playing L rounds.

We are ready to establish the connection between SWL and the (u, v)-pebbling game. Below, denote $\chi^{\mathcal{A},(t)}$ as the color mapping of SWL algorithm $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ at iteration t.

Lemma H.2. *Let $l \in \mathbb{N}$ be any integer. For any vertices $u_G, v_G \in \mathcal{V}_G$ and $u_H, v_H \in \mathcal{V}_H$, if $\chi_G^{\mathcal{A},(l)}(u_G, v_G) \neq \chi_H^{\mathcal{A},(l)}(u_H, v_H)$, then Spoiler can win the l-round (u, v)-pebbling game $\mathsf{G}^{\mathcal{A},l}(u; v)$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$.*

Proof. The proof is based on induction over l. First consider the base case of $l = 0$. If $\chi_G^{\mathcal{A},(0)}(u_G, v_G) \neq \chi_H^{\mathcal{A},(0)}(u_H, v_H)$, by definition of node marking policy we have either ($u_G = v_G$, $u_H \neq v_H$) or ($u_G \neq v_G$, $u_H = v_H$). Clearly, (u_G, v_G) and (u_H, v_H) have different isomorphism types and thus Spoiler wins.

Now assume that Lemma H.2 holds for $l \le L$, and consider $l = L + 1$. Let $\chi_G^{\mathcal{A},(L+1)}(u_G, v_G) \neq \chi_H^{\mathcal{A},(L+1)}(u_H, v_H)$. If $\chi_G^{\mathcal{A},(L)}(u_G, v_G) \neq \chi_H^{\mathcal{A},(L)}(u_H, v_H)$, then by induction Spoiler wins. Otherwise, there exists an aggregation operation $\mathsf{agg} \in \mathcal{A}$ such that

$$\mathsf{agg}(u_G, v_G, G, \chi_G^{\mathcal{A},(L)}) \neq \mathsf{agg}(u_H, v_H, H, \chi_H^{\mathcal{A},(L)}).$$

We separately consider which type of atomic aggregation operation agg is:

- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}}$. In this case, we have $\chi_G^{\mathcal{A},(L)}(v_G, u_G) \neq \chi_H^{\mathcal{A},(L)}(v_H, u_H)$. In the first round, Spoiler can choose to swap pebbles u and v. The remaining game will then be equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (v_G, v_H)$, $v = (u_G, u_H)$. By induction, Spoiler wins the game.
- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$. In this case, we have $\chi_G^{\mathcal{A},(L)}(u_G, u_G) \neq \chi_H^{\mathcal{A},(L)}(u_H, u_H)$. In the first round, Spoiler can choose to move pebbles v to the position of u. The remaining game will then be equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (u_G, u_H)$, $v = (u_G, u_H)$. By induction, Spoiler wins the game.
- Global aggregation $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$. In this case, we have

$$\{\!\{\chi_G^{\mathcal{A},(L)}(u_G, w_G) : w_G \in \mathcal{V}_G\}\!\} \neq \{\!\{\chi_H^{\mathcal{A},(L)}(u_H, w_H) : w_H \in \mathcal{V}_H\}\!\}.$$

Therefore, there exists a color c such that $|\mathcal{C}_G(u_G, c)| \neq |\mathcal{C}_H(u_H, c)|$, where we denote

$$\mathcal{C}_G(u_G, c) = |\{w_G \in \mathcal{V}_G : \chi_G^{\mathcal{A},(L)}(u_G, w_G) = c\}|.$$

If $|\mathcal{C}_G(u_G, c)| > |\mathcal{C}_H(u_H, c)|$, Spoiler can select the vertex subset $\mathcal{S}^{\mathsf{S}} = \mathcal{C}_G(u_G, c) \subset \mathcal{V}_G$. It can be seen that no matter how Duplicator responds with $\mathcal{S}^{\mathsf{D}} \subset \mathcal{V}_H$, there exists $w_H \in \mathcal{S}^{\mathsf{D}}$ such that $\chi_H^{\mathcal{A},(L)}(u_H, w_H) \neq c$. Spoiler thus select this vertex $x^{\mathsf{S}} = w_H$, and no matter how Duplicator responds with $x^{\mathsf{D}} = w_G \in \mathcal{S}^{\mathsf{S}}$, we have $\chi_G^{\mathcal{A},(L)}(u_G, w_G) \neq \chi_H^{\mathcal{A},(L)}(u_H, w_H)$. The remaining game will then be equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (u_G, u_H)$, $v = (w_G, w_H)$. By induction, Spoiler wins the game.

If $|\mathcal{C}_G(u_G, c)| < |\mathcal{C}_H(u_H, c)|$, Spoiler can select the vertex subset $\mathcal{S}^{\mathsf{S}} = \mathcal{C}_H(u_H, c) \subset \mathcal{V}_H$, and the conclusion is the same.

- Local aggregation $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}$. In this case, we have

$$\{\!\{\chi_G^{\mathcal{A},(L)}(u_G, w_G) : w_G \in \mathcal{N}_G(v_G)\}\!\} \neq \{\!\{\chi_H^{\mathcal{A},(L)}(u_H, w_H) : w_H \in \mathcal{N}_H(v_H)\}\!\}.$$

Therefore, there exists a color c such that $|\mathcal{C}_G(u_G, v_G, c)| \neq |\mathcal{C}_H(u_H, v_H, c)|$, where we denote

$$\mathcal{C}_G(u_G, v_G, c) = |\{w_G \in \mathcal{N}_G(v_G) : \chi_G^{\mathcal{A},(L)}(u_G, w_G) = c\}|.$$

If $|\mathcal{C}_G(u_G, v_G, c)| > |\mathcal{C}_H(u_H, v_H, c)|$, Spoiler can select the vertex subset $\mathcal{S}^{\mathsf{S}} = \mathcal{C}_G(u_G, v_G, c) \subset \mathcal{N}_G(v_G)$. If $|\mathcal{C}_G(u_G, v_G, c)| < |\mathcal{C}_H(u_H, v_H, c)|$, Spoiler can select the vertex subset $\mathcal{S}^{\mathsf{S}} = \mathcal{C}_H(u_H, v_H, c) \subset \mathcal{N}_H(v_H)$. Using a similar analysis as the above case, we can conclude that Spoiler wins the game.

The cases of $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$, $\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}$, $\mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}$ are similar (symmetric) to $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}$, $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$, $\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}$, so we omit them for clarity. We have concluded the induction step. □

Lemma H.3. *Let $l \in \mathbb{N}$ be any integer. Assume $\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\} \cap \mathcal{A} \neq \emptyset$. For any vertices $u_G, v_G \in \mathcal{V}_G$ and $u_H, v_H \in \mathcal{V}_H$, if $\chi_G^{\mathcal{A},(l+1)}(u_G, v_G) = \chi_H^{\mathcal{A},(l+1)}(u_H, v_H)$, then Duplicator can win the l-round (u, v)-pebbling game $\mathsf{G}^{\mathcal{A},l}(u; v)$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$.*

Proof. The proof is based on induction over l. First, consider the base case of $l = 0$. Let $\chi_G^{\mathcal{A},(1)}(u_G, v_G) = \chi_H^{\mathcal{A},(1)}(u_H, v_H)$. If $u_G = v_G$, then $u_H = v_H$ (due to the node marking policy). If $\{u_G, v_G\} \in \mathcal{E}_G$, then $\{u_H, v_H\} \in \mathcal{E}_H$ (which follows by applying the local aggregation, similar to the proof of Lemma E.4). Therefore, (u_G, v_G) and (u_H, v_H) have the same isomorphism type.

Now assume that Lemma H.3 holds for $l \leq L$, and consider $l = L + 1$. Let $\chi_G^{\mathcal{A},(L+2)}(u_G, v_G) = \chi_H^{\mathcal{A},(L+2)}(u_H, v_H)$. Then,

$$\mathsf{agg}(u_G, v_G, G, \chi_G^{\mathcal{A},(L+1)}) = \mathsf{agg}(u_H, v_H, H, \chi_H^{\mathcal{A},(L+1)}) \tag{20}$$

holds for all $\mathsf{agg} \in \mathcal{A}$. Separately consider various possible strategies for Spoiler:

- If $\mathsf{agg}_{\mathsf{vu}}^{\mathsf{P}} \in \mathcal{A}$ and Spoiler chooses to swap pebbles u and v. Setting $\mathsf{agg} = \mathsf{agg}_{\mathsf{vu}}^{\mathsf{P}}$ in (20) yields $\chi_G^{\mathcal{A},(L+1)}(v_G, u_G) = \chi_H^{\mathcal{A},(L+1)}(v_H, u_H)$. The remaining game is equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (v_G, v_H)$, $v = (u_G, u_H)$. By induction, Duplicator wins the game.
- If $\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}} \in \mathcal{A}$ and Spoiler chooses to move pebbles v to the position of pebble u. This case is similar to the above one, and we have $\chi_G^{\mathcal{A},(L+1)}(u_G, u_G) = \chi_H^{\mathcal{A},(L+1)}(u_H, u_H)$. The remaining game is equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (u_G, u_H)$, $v = (u_G, u_H)$. By induction, Duplicator wins the game.
- If $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}} \in \mathcal{A}$, and Spoiler chooses a subset $\mathcal{S}^{\mathsf{S}}$. Setting $\mathsf{agg} = \mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$ in (20) yields

$$\{\!\{\chi_G^{\mathcal{A},(L+1)}(u_G, w_G) : w_G \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H^{\mathcal{A},(L+1)}(u_H, w_H) : w_H \in \mathcal{V}_H\}\!\}.$$

If $\mathcal{S}^{\mathsf{S}} \subset \mathcal{V}_G$, then Duplicator can respond with a subset $\mathcal{S}^{\mathsf{D}} \subset \mathcal{V}_H$ such that

$$\{\!\{\chi_G^{\mathcal{A},(L+1)}(u_G, w_G) : w_G \in \mathcal{S}^{\mathsf{S}}\}\!\} = \{\!\{\chi_H^{\mathcal{A},(L+1)}(u_H, w_H) : w_H \in \mathcal{S}^{\mathsf{D}}\}\!\}.$$

If $\mathcal{S}^{\mathsf{S}} \subset \mathcal{V}_H$, then Duplicator can respond with a subset $\mathcal{S}^{\mathsf{D}} \subset \mathcal{V}_G$ such that

$$\{\!\{\chi_G^{\mathcal{A},(L+1)}(u_G, w_G) : w_G \in \mathcal{S}^{\mathsf{D}}\}\!\} = \{\!\{\chi_H^{\mathcal{A},(L+1)}(u_H, w_H) : w_H \in \mathcal{S}^{\mathsf{S}}\}\!\}.$$

In both cases, we clearly have $|\mathcal{S}^{\mathsf{S}}| = |\mathcal{S}^{\mathsf{D}}|$. Next, no matter how Spoiler moves pebble v to a vertex $x^{\mathsf{S}} \in \mathcal{S}^{\mathsf{D}}$, Duplicator can always respond by moving the other pebble v to a vertex $x^{\mathsf{D}} \in \mathcal{S}^{\mathsf{S}}$, such that $\chi_G^{\mathcal{A},(L+1)}(u_G, \tilde{v}_G) = \chi_H^{\mathcal{A},(L+1)}(u_H, \tilde{v}_H)$, where $(\tilde{v}_G, \tilde{v}_H)$ is the new position of pebbles v. The remaining game is equivalent to $\mathsf{G}^{\mathcal{A},L}(u, v)$ with $u = (u_G, u_H)$, $v = (\tilde{v}_G, \tilde{v}_H)$. By induction, Duplicator wins the game.

- If $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$, then all the procedure is similar to the above one except that the subsets $\mathcal{S}^{\mathsf{S}}$ and $\mathcal{S}^{\mathsf{D}}$ contain only the neighboring vertices adjacent to pebbles v.

The cases of $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$, $\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}$, $\mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}$ are similar (symmetric) to $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}$, $\mathsf{agg}_{\mathsf{u}}^{\mathsf{G}}$, $\mathsf{agg}_{\mathsf{uu}}^{\mathsf{P}}$, so we omit them for clarity. We have concluded the induction step. □

Combining Lemmas H.2 and H.3 immediately yields the following theorem:

Theorem H.4. *Let $\chi^{\mathcal{A}}$ be the stable color mapping of SWL algorithm $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$, satisfying $\{\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}\} \cap \mathcal{A} \neq \emptyset$. For any vertices $u_G, v_G \in \mathcal{V}_G$ and $u_H, v_H \in \mathcal{V}_H$, $\chi_G^{\mathcal{A}}(u_G, v_G) = \chi_H^{\mathcal{A}}(u_H, v_H)$ if and only if Duplicator can win the l-round (u, v)-pebbling game $\mathsf{G}^{\mathcal{A},l}(u; v)$ for any $l \in \mathbb{N}$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$.*

We next turn to FWL-type algorithms. We can similarly define the L-round (u, v)-pebbling game G^L for $\mathsf{FWL}(2)$, $\mathsf{LFWL}(2)$, and $\mathsf{SLFWL}(2)$. We have the following theorem parallel to Theorem H.4.

Theorem H.5. *Let χ be the stable color mapping of any FWL-type algorithm, e.g., $\mathsf{FWL}(2)$, $\mathsf{LFWL}(2)$, and $\mathsf{SLFWL}(2)$. For any vertices $u_G, v_G \in \mathcal{V}_G$ and $u_H, v_H \in \mathcal{V}_H$, $\chi_G(u_G, v_G) = \chi_H(u_H, v_H)$ if and only if Duplicator can win the corresponding l-round (u, v)-pebbling game $\mathsf{G}^l(u; v)$ for any $l \in \mathbb{N}$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$.*

Proof. The proof is highly similar to the proof of Lemmas H.2 and H.3. For clarity, we only take $\mathsf{LFWL}(2)$ as an example. We use induction over l to prove that for any $l \in \mathbb{N}$, $\chi_G^{(l)}(u_G, v_G) = \chi_H^{(l)}(u_H, v_H)$ if and only if Duplicator can win the l-round (u, v)-pebbling game $\mathsf{G}^l(u; v)$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$. The base case of $l = 0$ is trivial.

For the induction step, suppose the result holds for $l \leq L$ and consider $l = L + 1$. Let $\chi_G^{(L+1)}(u_G, v_G) \neq \chi_H^{(L+1)}(u_H, v_H)$. If $\chi_G^{(L)}(u_G, v_G) \neq \chi_H^{(L)}(u_H, v_H)$, Spoiler wins by induction. Otherwise, by definition of $\mathsf{LFWL}(2)$ we have

$$\{\!\{(\chi_G^{(L)}(u_G, w_G), \chi_G^{(L)}(w_G, v_G)) : w_G \in \mathcal{N}_G^1(v_G)\}\!\} \neq \{\!\{(\chi_H^{(L)}(u_H, w_H), \chi_H^{(L)}(w_H, v_H)) : w_H \in \mathcal{N}_H^1(v_H)\}\!\}.$$

Therefore, there exists a color c such that $|\mathcal{C}_G(u_G, v_G, c)| \neq |\mathcal{C}_H(u_H, v_H, c)|$, where we denote

$$\mathcal{C}_G(u, v, c) = |\{w \in \mathcal{N}_G^1(v) : (\chi_G^{(L)}(u, w), \chi_G^{(L)}(w, v)) = c\}|.$$

Assume $|\mathcal{C}_G(u_G, v_G, c)| > |\mathcal{C}_H(u_H, v_H, c)|$ without loss of generality, then Spoiler can select the vertex subset $\mathcal{S}^{\mathsf{S}} = \mathcal{C}_G(u_G, v_G, c) \subset \mathcal{N}_G^1(v_G)$. No matter how Duplicator responds with $\mathcal{S}^{\mathsf{D}} \subset \mathcal{N}_H^1(v_H)$, there exists $w_H \in \mathcal{S}^{\mathsf{D}}$ such that $(\chi_H^{(L)}(u_H, w_H), \chi_H^{(L)}(w_H, v_H)) \neq c$. Spoiler thus select this vertex $x^{\mathsf{S}} = w_H$, and no matter how Duplicator responds with $x^{\mathsf{D}} = w_G \in \mathcal{S}^{\mathsf{S}}$, we have either $\chi_G^{(L)}(u_G, w_G) \neq \chi_H^{(L)}(u_H, w_H)$ or $\chi_G^{(L)}(w_G, v_G) \neq \chi_H^{(L)}(w_H, v_H)$. Spoiler chooses to move pebbles v or u depending on which relation does not hold. The remaining game will then be equivalent to $\mathsf{G}^L(u, v)$ with $u = (\tilde{u}_G, \tilde{u}_H)$, $v = (\tilde{v}_G, \tilde{v}_H)$ such that $\chi_G^{(L)}(\tilde{u}_G, \tilde{v}_G) \neq \chi_H^{(L)}(\tilde{u}_H, \tilde{v}_H)$. By induction, Spoiler wins the game.

For the converse direction, the proof is similar and we omit it for clarity. □

Finally, we complete the analysis by incorporating different pooling paradigms into pebbling games. We will prove the following general result:

Lemma H.6. *Let χ be the stable color mapping of any WL algorithm and let G be the corresponding pebbling game, such that $\chi_G(u_G, v_G) = \chi_H(u_H, v_H)$ if and only if Duplicator can win the l-round (u, v)-pebbling game $\mathsf{G}^l(u; v)$ for all $l \in \mathbb{N}$ with $u = (u_G, u_H)$, $v = (v_G, v_H)$. Then,*

- $\{\!\{\chi_G(u_G, v_G) : u_G, v_G \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(u_H, v_H) : u_H, v_H \in \mathcal{V}_H\}\!\}$ *if and only if Duplicator can win the pebbling game when* $u = (u_G, u_H)$, $v = (v_G, v_H)$ *are selected according to the game rule of FWL-type algorithms defined in Section 6;*
- $\{\!\{\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} : u_G \in \mathcal{V}_G\}\!\} = \{\!\{\{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\} : u_H \in \mathcal{V}_H\}\!\}$ *if and only if Duplicator can win the pebbling game when* $u = (u_G, u_H)$, $v = (v_G, v_H)$ *are selected according to the game rule of* VS *pooling defined in Section 6;*
- $\{\!\{\{\!\{\chi_G(u_G, v_G) : u_G \in \mathcal{V}_G\}\!\} : v_G \in \mathcal{V}_G\}\!\} = \{\!\{\{\!\{\chi_H(u_H, v_H) : u_H \in \mathcal{V}_H\}\!\} : v_H \in \mathcal{V}_H\}\!\}$ *if and only if Duplicator can win the pebbling game when* $u = (u_G, u_H)$, $v = (v_G, v_H)$ *are selected according to the game rule of* SV *pooling defined in Section 6.*

Proof. We only prove the second bullet and other cases are similar. First assume $\{\!\{\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} : u_G \in \mathcal{V}_G\}\!\} = \{\!\{\{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\} : u_H \in \mathcal{V}_H\}\!\}$. According to the game rule, both players first place pebbles u based on a vertex selection procedure. Without loss of generality, suppose Spoiler chooses a subset $\mathcal{S}^{\mathsf{S}} \subset \mathcal{V}_G$. Then Duplicator can respond with a subset $\mathcal{S}^{\mathsf{D}} \subset \mathcal{V}_H$ such that

$$\{\!\{\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} : u_G \in \mathcal{S}^{\mathsf{S}}\}\!\} = \{\!\{\{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\} : u_H \in \mathcal{S}^{\mathsf{D}}\}\!\}.$$

Then no matter how Spoiler selects a vertex $x^{\mathsf{S}} = u_H \in \mathcal{S}^{\mathsf{D}}$, Duplicator can always select $x^{\mathsf{D}} = u_G \in \mathcal{S}^{\mathsf{S}}$, such that

$$\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\}.$$

Similarly, after selecting the position of pebbles v, Duplicator always has a strategy to ensure that $\chi_G(u_G, v_G) = \chi_H(u_H, v_H)$. For the remaining game, Duplicator can win due to the assumption of Lemma H.6.

For the converse direction, assume $\{\!\{\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} : u_G \in \mathcal{V}_G\}\!\} \neq \{\!\{\{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\} : u_H \in \mathcal{V}_H\}\!\}$. Similar to the proof of global aggregation in Lemma H.2, Spoiler has a strategy to ensure that

$$\{\!\{\chi_G(u_G, v_G) : v_G \in \mathcal{V}_G\}\!\} \neq \{\!\{\chi_H(u_H, v_H) : v_H \in \mathcal{V}_H\}\!\}$$

after placing pebble u to position (u_G, u_H). Again, after placing pebble v to position (v_G, v_H), Spoiler has a strategy to ensure that $\chi_G(u_G, v_G) \neq \chi_H(u_H, v_H)$. For the remaining game, Spoiler can win due to the assumption of Lemma H.6. □

Consequently, Theorem 6.2 and Theorem 6.3 holds by Theorems H.4 and H.5 and Lemma H.6.

I. Proof of Separation Results (Theorem 7.1)

This section contains the proof of the main result in this paper (Theorem 7.1). The proof is quite involved and is divided into three parts. First, we introduce a novel construction of counterexample graphs that are based on (and greatly extend) the work of Fürer (2001). We provide an in-depth analysis of the isomorphism properties of these counterexample graphs through a set of key theorems. Then, in light of the special properties, we simplify the pebbling game developed in Section 6 for each type of SWL/FWL algorithm, which specifically targets these counterexample graphs. Finally, we prove all separation results in Section 7 using the pebbling game viewpoint and give concrete counterexample graphs for each pair of algorithms.

I.1. Generalized Fürer graphs and their properties

We first introduce a class of graphs which we call the Fürer graphs (Fürer, 2001).

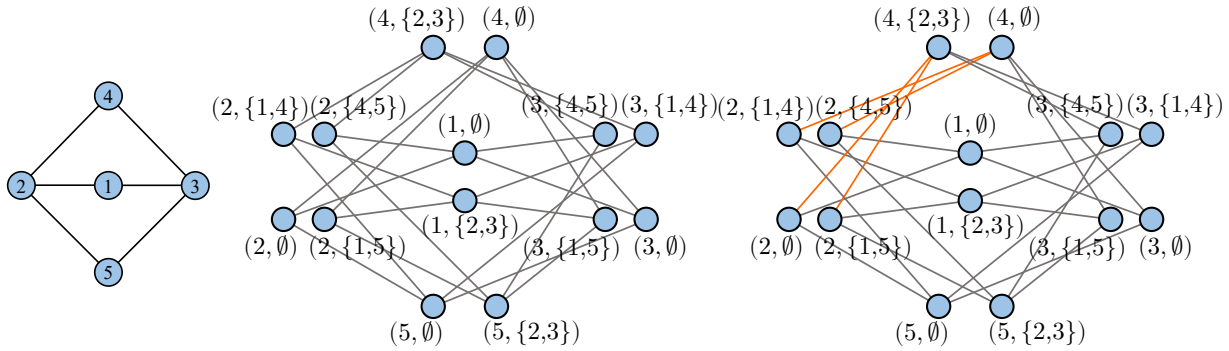
Definition I.1 (Fürer graphs)**.** Given any connected graph $F = (\mathcal{V}_F, \mathcal{E}_F)$, the Fürer graph $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ is constructed as follows:

$$\mathcal{V}_G = \{(x, \mathcal{X}) : x \in \mathcal{V}_F, \mathcal{X} \subset \mathcal{N}_F(x), |\mathcal{X}| \bmod 2 = 0\},$$
$$\mathcal{E}_G = \{\{(x, \mathcal{X}), (y, \mathcal{Y})\} \subset \mathcal{V}_G : \{x, y\} \in \mathcal{E}_F, (x \in \mathcal{Y} \leftrightarrow y \in \mathcal{X})\}.$$

Here, $x \in \mathcal{Y} \leftrightarrow y \in \mathcal{X}$ means that either ($x \in \mathcal{Y}$ and $y \in \mathcal{X}$) or ($x \notin \mathcal{Y}$ and $y \notin \mathcal{X}$). For each vertex $x \in \mathcal{V}_F$, denote the set

$$\mathsf{Meta}_F(x) := \{(x, \mathcal{X}) : \mathcal{X} \subset \mathcal{N}_F(x), |\mathcal{X}| \bmod 2 = 0\}, \tag{21}$$

which is called the meta vertices of $G(F)$ associated to vertex F. Clearly, $\mathcal{V}_G = \bigcup_{x \in \mathcal{V}_F} \mathsf{Meta}_F(x)$.



(a) Base graph F (b) Fürer graph $G(F)$ (c) Twisted Fürer graph $H(F)$ for edge $\{2, 4\}$

Figure 2. Illustration of the construction of Fürer graph and twisted Fürer graph.

We next define an operation called "twist":

Definition I.2 (Twist)**.** Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$, and let $\{x, y\} \in \mathcal{E}_F$ be an edge of F. The *twisted* Fürer graph for edge $\{x, y\}$ is constructed as follows: $\mathsf{twist}(G(F), \{x, y\}) := (\mathcal{V}_G, \mathcal{E}_H)$, where

$$\mathcal{E}_H := \mathcal{E}_G \triangle \{\{\xi, \eta\} : \xi \in \mathsf{Meta}_F(x), \eta \in \mathsf{Meta}_F(y)\}.$$

Here, $\triangle$ is the symmetric difference operation, i.e., $\mathcal{A} \triangle \mathcal{B} = (\mathcal{A} \backslash \mathcal{B}) \cup (\mathcal{B} \backslash \mathcal{A})$.

In other words, the twisted Fürer graph $\mathsf{twist}(G(F), \{x, y\})$ is the graph modified from $G(F)$ by deleting all edges of the form $\{(x, \mathcal{X}), (y, \mathcal{Y})\} \in \mathcal{E}_G$ and adding the following set of edges

$$\{\{(x, \mathcal{X}), (y, \mathcal{Y})\} \subset \mathcal{V}_G : (x \in \mathcal{Y} \leftrightarrow y \notin \mathcal{X})\}.$$

We give an illustration of the construction of Fürer graph and twisted Fürer graph for a simple graph F in Figure 2.

The twist operation can be further generalized into twisting a set of edges. We adopt the following notations:

$$\mathsf{twist}(G(F), \mathcal{E}) := \mathsf{twist}(\cdots \mathsf{twist}(G(F), e_1) \cdots, e_k) \tag{22}$$

given an edge set $\mathcal{E} = \{e_1, \cdots, e_k\} \subset \mathcal{E}_F$. Note that the resulting graph $\mathsf{twist}(G(F), \mathcal{E})$ does not depend on the order of edges $e_1, \cdots, e_k$ for twisting, so (22) is well-defined.

The first key result below shows that if we twist any two edges of a Fürer graph, the resulting graph is isomorphic to the original graph.

Lemma I.3. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$. Then, for any two different edges $\{x, y\}, \{u, v\} \in \mathcal{E}_F$,*

$$\mathsf{twist}(G(F), \{\{x, y\}, \{u, v\}\}) \simeq G(F).$$

Moreover, there exists an isomorphism $f : \mathcal{V}_G \to \mathcal{V}_G$ from $G(F)$ to $\mathsf{twist}(G(F), \{\{x, y\}, \{u, v\}\})$ that maps each meta vertex set $\mathsf{Meta}_F(x)$ to itself for all $x \in \mathcal{V}_F$.

Proof. Denote $\widehat{G}(F) := \mathsf{twist}(G(F), \{\{x, y\}, \{u, v\}\})$. Since F is connected, one can always find a simple path $(w_0, w_1, \cdots, w_k)$, $k \geq 1$ with $\{w_0, w_1\} = \{x, y\}$ and $\{w_{k-1}, w_k\} = \{u, v\}$. Denote $\mathcal{P} = \{w_1, \cdots, w_{k-1}\}$. Construct a mapping $f : \mathcal{V}_G \to \mathcal{V}_G$ as follows:

$$f(z, \mathcal{Z}) = \begin{cases} (z, \mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}) & \text{if } z = w_i, i \in [k-1], \\ (z, \mathcal{Z}) & \text{if } z \notin \mathcal{P}. \end{cases} \tag{23}$$

We will prove that f is an isomorphism from $G(F)$ to $\widehat{G}(F)$. First, since $|\mathcal{Z}| \bmod 2 = 0$ implies that $|\mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}| \bmod 2 = 0$, f is indeed a valid mapping from $\mathcal{V}_G$ to $\mathcal{V}_G$. Also, it is straightforward to see that f is bijective. It remains to verify that for any edge $\{(z, \mathcal{Z}), (z', \mathcal{Z}')\} \in \mathcal{E}_G$, $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\}$ is an edge of $\widehat{G}(F)$. Separately consider the following cases:

- If $z, z' \notin \mathcal{P}$, then $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\} = \{(z, \mathcal{Z}), (z', \mathcal{Z}')\}$ is clearly an edge of $\widehat{G}(F)$.
- If $z, z' \in \mathcal{P}$, denote $z = w_i$ and $z' = w_j$. Then it is straightforward to see that $w_i \in \mathcal{Z}' \leftrightarrow w_j \in \mathcal{Z}$ if and only if $w_i \in \mathcal{Z}' \triangle \{w_{j-1}, w_{j+1}\} \leftrightarrow w_j \in \mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}$. Therefore, $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\} = \{(z, \mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}), (z', \mathcal{Z}' \triangle \{w_{j-1}, w_{j+1}\})\}$ is an edge of $\widehat{G}(F)$.
- If $z = w_i \in \mathcal{P}$, $z' \notin \mathcal{P}$, $\{z, z'\} \neq \{x, y\}$, and $\{z, z'\} \neq \{u, v\}$, then $z' \neq w_{i-1}$ and $z' \neq w_{i+1}$. Therefore, $z \in \mathcal{Z}' \leftrightarrow z' \in \mathcal{Z}$ if and only if $z \in \mathcal{Z}' \leftrightarrow z' \in \mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}$. This implies that $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\} = \{(z, \mathcal{Z} \triangle \{w_{i-1}, w_{i+1}\}), (z', \mathcal{Z}')\}$ is an edge of $\widehat{G}(F)$.
- If $\{z, z'\} = \{x, y\}$, we can denote $z = w_0$ and $z' = w_1$. We have $z \in \mathcal{Z}' \leftrightarrow z' \in \mathcal{Z}$ if and only if $z \notin \mathcal{Z}' \triangle \{w_0, w_2\} \leftrightarrow z' \in \mathcal{Z}$. Note that there is a twist in $\widehat{G}(F)$ for edge $\{x, y\}$, so we still obtain that $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\} = \{(z, \mathcal{Z}), (z', \mathcal{Z}' \triangle \{w_0, w_2\})\}$ is an edge of $\widehat{G}(F)$.
- Finally, if $\{z, z'\} = \{u, v\}$, the analysis is the same as the above one and $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\}$ is an edge of $\widehat{G}(F)$.

In all cases, $\{f(z, \mathcal{Z}), f(z', \mathcal{Z}')\}$ is an edge of $\widehat{G}(F)$. Moreover, it is clear that f maps each meta vertex set $\mathsf{Meta}_F(x)$ to itself for all $x \in \mathcal{V}_F$, which concludes the proof. □

Based on Lemma I.3, it is convenient to define a notion called *proper* isomorphism:

Definition I.4 (Proper isomorphism)**.** Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$ and $\widehat{G}(F) = \mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E} \subset \mathcal{E}_F$. We say f is a proper isomorphism from $G(F)$ to $\widehat{G}(F)$, if f is an isomorphism from $G(F)$ to $\widehat{G}(F)$ that maps each meta vertex set $\mathsf{Meta}_F(x)$ to itself for all $x \in \mathcal{V}_F$.

Lemma I.3 can be generalized into the following corollary:

Corollary I.5. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$. Then, for any edge set $\mathcal{E} \subset \mathcal{E}_F$ and any two different edges $\{x, y\}, \{u, v\} \in \mathcal{E}_F$,*

$$\mathsf{twist}(G(F), \mathcal{E} \triangle \{\{x, y\}, \{u, v\}\}) \simeq \mathsf{twist}(G(F), \mathcal{E}).$$

Moreover, any proper isomorphism $f : \mathcal{V}_G \to \mathcal{V}_G$ from $G(F)$ to $\mathsf{twist}(G(F), \{\{x, y\}, \{u, v\}\})$ is also a proper isomorphism from $\mathsf{twist}(G(F), \mathcal{E})$ to $\mathsf{twist}(G(F), \mathcal{E} \triangle \{\{x, y\}, \{u, v\}\})$.

Proof. Denote $\widehat{G}(F) := \mathsf{twist}(G(F), \{\{u, v\}, \{x, y\}\})$, $H(F) := \mathsf{twist}(G(F), \mathcal{E})$, and $\widehat{H}(F) := \mathsf{twist}(\widehat{G}(F), \mathcal{E})$. Note that by definition of the twist operation, we equivalently have $\widehat{H}(F) = \mathsf{twist}(G(F), \mathcal{E} \triangle \{\{x, y\}, \{u, v\}\})$. Due to Lemma I.3, we have $\widehat{G}(F) \simeq G(F)$. Let f be a proper isomorphism from $G(F)$ to $\widehat{G}(F)$ (according to Lemma I.3). It suffices to prove that f is also an isomorphism from $H(F)$ to $\widehat{H}(F)$.

For any edge $\{(w, \mathcal{W}), (z, \mathcal{Z})\}$ in $H(F)$:

- If $\{w, z\} \in \mathcal{E}$, then $\{(w, \mathcal{W}), (z, \mathcal{Z})\}$ is not an edge in $G(F)$. Therefore, $\{f(w, \mathcal{W}), f(z, \mathcal{Z})\}$ is not an edge in $\widehat{G}(F)$. Since f maps $\mathsf{Meta}_F(w)$ to $\mathsf{Meta}_F(w)$ and maps $\mathsf{Meta}_F(z)$ to $\mathsf{Meta}_F(z)$, we obtain that $\{f(w, \mathcal{W}), f(z, \mathcal{Z})\}$ is an edge in $\widehat{H}(F)$.
- If $\{w, z\} \notin \mathcal{E}$, then $\{(w, \mathcal{W}), (z, \mathcal{Z})\}$ is an edge in $G(F)$. Therefore, $\{f(w, \mathcal{W}), f(z, \mathcal{Z})\}$ is an edge in $\widehat{G}(F)$. Since f maps $\mathsf{Meta}_F(w)$ to $\mathsf{Meta}_F(w)$ and maps $\mathsf{Meta}_F(z)$ to $\mathsf{Meta}_F(z)$, we also obtain that $\{f(w, \mathcal{W}), f(z, \mathcal{Z})\}$ is an edge in $\widehat{H}(F)$.

In both cases, $\{f(w, \mathcal{W}), f(z, \mathcal{Z})\}$ is an edge in $\widehat{H}(F)$. Since Lemma I.3 has proved that f is bijective, f is an isomorphism from $H(F)$ to $\widehat{H}(F)$ and thus $H(F) \simeq \widehat{H}(F)$. □

As a special case, Corollary I.5 leads to the following important fact:

Corollary I.6. *Let $G(F)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$. Then, for any two edges $\{x, y\}, \{u, v\} \in \mathcal{E}_F$,*

$$\mathsf{twist}(G(F), \{x, y\}) \simeq \mathsf{twist}(G(F), \{u, v\}).$$

Proof. Setting $\mathcal{E} = \{u, v\}$ in Corollary I.5 readily concludes the proof. □

Corollary I.6 shows that the structure of a twisted Fürer graph does not depend on which edge is twisted. Therefore, we can simply denote $H(F)$ as the twisted Fürer graph of F without specifying the twisted edge $\{x, y\}$. Moreover, recursively applying Corollary I.5 obtains that, if we twist k edges of a Fürer graph $G(F)$, the resulting graph is isomorphic to either $G(F)$ or $H(F)$, depending on whether k is even or odd. To complete the result, we show $G(F)$ and $H(F)$ are actually non-isomorphic under certain conditions:

Lemma I.7. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be the Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$, and let $H(F) = (\mathcal{V}_G, \mathcal{E}_H)$ be the twisted Fürer graph. Then there does not exist a proper isomorphism $f : \mathcal{V}_G \to \mathcal{V}_G$ from $G(F)$ to $H(F)$.*

Proof. Assume $H(F) = \mathsf{twist}(G(F), \{u, v\})$ for some $\{u, v\} \in \mathcal{E}_F$ and $f : \mathcal{V}_G \to \mathcal{V}_G$ is a proper isomorphism. Then we can write $f(x, \emptyset) = (x, \mathcal{T}_x)$ for all $x \in \mathcal{V}_F$. Note that for any $\{x, y\} \in \mathcal{E}_F$, by definition of the Fürer graph there is an edge between vertices $(x, \emptyset)$ and $(y, \emptyset)$ in $G(F)$. However, we will prove that this is not the case for $H(F)$: there must exist an odd number of edges $\{x, y\} \in \mathcal{E}_F$ such that $\{(x, \mathcal{T}_x), (y, \mathcal{T}_y)\} \notin \mathcal{E}_H$. This will lead to a contradiction and finish the proof.

Formally, let $\mathcal{E} = \{\{(x, \mathcal{T}_x), (y, \mathcal{T}_y)\} : \{x, y\} \in \mathcal{E}_F\}$, and our goal is prove that $|\mathcal{E} \backslash \mathcal{E}_H| \bmod 2 = 1$. The proof is based on induction. First, consider the base case when $\mathcal{T}_x = \emptyset$ for all $x \in \mathcal{V}_F$. Clearly, there is exactly one element $\{(u, \mathcal{T}_u), (v, \mathcal{T}_v)\} \notin \mathcal{E}_H$ since $H(F)$ is obtained from $G(F)$ by twisting edge $\{u, v\}$. Next, for the induction step, we show if $|\mathcal{E} \backslash \mathcal{E}_H| \bmod 2 = 1$, then $|\tilde{\mathcal{E}} \backslash \mathcal{E}_H| \bmod 2 = 1$ holds for any $\tilde{\mathcal{E}}$ that is modified from $\mathcal{E}$ by changing a given $\mathcal{S}_z$ to another feasible $\tilde{\mathcal{S}}_z$ for some $z \in \mathcal{V}_F$, namely,

$$\tilde{\mathcal{E}} = \{\{(x, \mathcal{T}_x), (y, \mathcal{T}_y)\} : \{x, y\} \in \mathcal{E}_F, x, y \neq z\} \cup \{\{(x, \mathcal{T}_x), (z, \tilde{\mathcal{T}}_z)\} : \{x, z\} \in \mathcal{E}_F\}.$$

This is because

$$\begin{aligned} |\tilde{\mathcal{E}} \backslash \mathcal{E}_H| - |\mathcal{E} \backslash \mathcal{E}_H| &\equiv |(\tilde{\mathcal{E}} \triangle \mathcal{E}) \backslash \mathcal{E}_H| \\ &\equiv |\{x \in \mathcal{N}_F(z) : x \in \mathcal{T}_z \leftrightarrow x \notin \tilde{\mathcal{T}}_z\}| \\ &\equiv |\mathcal{T}_z \triangle \tilde{\mathcal{T}}_z| \equiv 0 \pmod 2, \end{aligned}$$

where the last equation holds because $|\mathcal{T}_z| \equiv |\tilde{\mathcal{T}}_z| \equiv 0 \pmod 2$. This concludes the induction step.

Since any set $\mathcal{E}$ can be obtained from the initial set $\{\{(x, \emptyset), (y, \emptyset)\} : \{x, y\} \in \mathcal{E}_F\}$ by modifying $\emptyset$ to $\mathcal{T}_z$ for each $z \in \mathcal{V}_F$ and the parity of $|\mathcal{E} \backslash \mathcal{E}_H|$ does not change throughout the process, we have concluded the proof. □

Below, we proceed to perform an in-depth analysis of the properties regarding the isomorphisms of (twisted) Fürer graphs. We need several definitions.

Definition I.8 (Connected components)**.** Let $F = (\mathcal{V}_F, \mathcal{E}_F)$ be a connected graph and let $\mathcal{S} \subset \mathcal{V}_F$ be a vertex set, called separation vertices. We say two edges $\{u, v\}, \{x, y\} \in \mathcal{E}_F$ are in the same connected component if there is a path $(y_0, y_1, \cdots, y_k)$ satisfying that $\{y_0, y_1\} = \{u, v\}$, $\{y_{k-1}, y_k\} = \{x, y\}$ and $y_i \notin \mathcal{S}$ for all $i \in [k-1]$. It is easy to see that the above relationship between edges forms an *equivalence relation*. Therefore, we can define a partition over the edge set: $\mathsf{CC}_{\mathcal{S}}(F) = \{\mathcal{P}_i : i \in [M]\}$, where each $\mathcal{P}_i \subset \mathcal{E}_F$ is called a connected component.

We are ready to state the central theorem:

Theorem I.9. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be either the original or twisted Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$, and let $\mathcal{S} \subset \mathcal{V}_F$ be any set. For each $u \in \mathcal{S}$, let $(u, \mathcal{T}_u), (u, \mathcal{U}_u) \in \mathsf{Meta}_F(u)$ be any given vertex sets. Then, there exists a proper isomorphism f from graph $G(F)$ to graph $\mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E} \subset \mathcal{E}_F$ with $|\mathcal{E}| \bmod 2 = 0$, such that $f(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. Moreover, for any $\tilde{\mathcal{E}} \subset \mathcal{E}_F$, there exists a proper isomorphism $\tilde{f}$ from $G(F)$ to $\mathsf{twist}(G(F), \tilde{\mathcal{E}})$ such that $\tilde{f}(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$ if and only if $|\mathcal{P} \cap \tilde{\mathcal{E}}| \equiv |\mathcal{P} \cap \mathcal{E}| \pmod 2$ for all $\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)$.*

Proof. We only consider the case when $G(F)$ is a Fürer graph, and the case of twisted Fürer graph is similar. We prove the theorem by induction over the size of $\mathcal{S}$. For the base case of $|\mathcal{S}| = 0$, there is clearly a trivial isomorphism (identity map) from $G(F)$ to $\mathsf{twist}(G(F), \emptyset) = G(F)$.

Now assume that the result holds for set $\mathcal{S}$, and there exists a proper isomorphism f from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E}$ with $|\mathcal{E}| \bmod 2 = 0$, such that $f(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. Consider adding a vertex $v \in \mathcal{V}_F$ and two given sets $(v, \mathcal{T}_v), (v, \mathcal{U}_v) \in \mathsf{Meta}_F(v)$. We will construct a new proper isomorphism f^{new} from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E}^{\mathsf{new}})$ for some

$\mathcal{E}^{\mathsf{new}} \subset \mathcal{E}_F$ with $|\mathcal{E}^{\mathsf{new}}| \bmod 2 = 0$, such that $f^{\mathsf{new}}(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$ and $f^{\mathsf{new}}(v, \mathcal{T}_v) = (v, \mathcal{U}_v)$. Denote $\tilde{\mathcal{U}}_v := f(v, \mathcal{T}_v)$ and denote $\mathcal{D}_v = \mathcal{U}_v \triangle \tilde{\mathcal{U}}_v$. Note that the size of $\mathcal{D}_v$ is even.

Denote $\mathcal{D}_v = \{x_1, \cdots, x_{2k}\}$. We define k mappings $f_i : \mathcal{V}_G \to \mathcal{V}_G$, $i \in [k]$ as follows:

$$f_i(w, \mathcal{W}) = \begin{cases} (w, \mathcal{W} \triangle \{x_{2i-1}, x_{2i}\}) & \text{if } w = v, \\ (w, \mathcal{W}) & \text{otherwise.} \end{cases} \tag{24}$$

We set f^{new} to be the composition of a series of mappings $f^{\mathsf{new}} := f_k \circ \cdots \circ f_1 \circ f$. By definition, we have

$$f^{\mathsf{new}}(v, \mathcal{T}_v) = (f_k \circ \cdots \circ f_1)(v, \tilde{\mathcal{U}}_v) = (v, \tilde{\mathcal{U}}_v \triangle \mathcal{D}_v) = (v, \mathcal{U}_v),$$

and for all $u \in \mathcal{S}$,

$$f^{\mathsf{new}}(u, \mathcal{T}_u) = (f_k \circ \cdots \circ f_1)(u, \mathcal{U}_u) = (u, \mathcal{U}_u).$$

It remains to verify that f^{new} is an isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E}^{\mathsf{new}})$ for some $\mathcal{E}^{\mathsf{new}}$ with $|\mathcal{E}^{\mathsf{new}}| \bmod 2 = 0$. Denote $\mathcal{E}^{(i)} = \bigcup_{j=1}^{2i} \{\{v, u_j\}\}$ for $i \in \{0, 1, \cdots, k\}$. Based on the proof of Lemma I.3 (i.e., the construction in (23)), f_i is an isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \{\{v, u_{2i-1}\}, \{v, u_{2i}\}\})$. Further using Corollary I.5, f_i is also an isomorphism from $\mathsf{twist}(G(F), \mathcal{E} \triangle \mathcal{E}^{(i-1)})$ to $\mathsf{twist}(G(F), \mathcal{E} \triangle \mathcal{E}^{(i)})$. Thus by composition, f^{new} is an isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E} \triangle \mathcal{E}^{(k)})$, namely, $\mathcal{E}^{\mathsf{new}} = \mathcal{E} \triangle \mathcal{E}^{(k)}$. Also, since $|\mathcal{E}^{(k)}| = 2k$ and $|\mathcal{E}| \bmod 2 = 0$, we have $|\mathcal{E}^{\mathsf{new}}| \bmod 2 = 0$, as desired. This finishes the induction step and concludes the proof of the first part.

For the second part, let us first consider how to find a proper isomorphism $\tilde{f}$ from $G(F)$ to $\mathsf{twist}(G(F), \tilde{\mathcal{E}})$ for other $\tilde{\mathcal{E}}$ satisfying $|\mathcal{P} \cap \tilde{\mathcal{E}}| \equiv |\mathcal{P} \cap \mathcal{E}| \pmod 2$ for all $\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)$, such that $\tilde{f}(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. For each $\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)$, denote $\mathcal{D}_{\mathcal{P}} := (\mathcal{E} \triangle \tilde{\mathcal{E}}) \cap \mathcal{P}$. Clearly, $\bigcup_{\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)} \mathcal{D}_{\mathcal{P}} = \mathcal{E} \triangle \tilde{\mathcal{E}}$. By assumption we have

$$\begin{aligned} |\mathcal{D}_{\mathcal{P}}| &= |(\mathcal{E} \triangle \tilde{\mathcal{E}}) \cap \mathcal{P}| = |(\mathcal{E} \cap \mathcal{P}) \triangle (\tilde{\mathcal{E}} \cap \mathcal{P})| \\ &\equiv |\mathcal{E} \cap \mathcal{P}| + |\tilde{\mathcal{E}} \cap \mathcal{P}| \equiv 0 \pmod 2. \end{aligned}$$

Therefore, there is a proper isomorphism $f^{\mathcal{P}}$ from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{D}_{\mathcal{P}})$ based on Lemma I.3 and Corollary I.5. Concretely, denote $\mathcal{D}_{\mathcal{P}} = \{e_1, \cdots, e_{2k}\}$, then $f^{\mathcal{P}}$ can be constructed as a composition $f^{\mathcal{P}} = f_k^{\mathcal{P}} \circ \cdots \circ f_1^{\mathcal{P}}$ where each $f_i^{\mathcal{P}}$ is a proper isomorphism from $\mathsf{twist}(G(F), \bigcup_{j=1}^{2(i-1)} \{e_j\})$ to $\mathsf{twist}(G(F), \bigcup_{j=1}^{2i} \{e_j\})$. Since all edges $e_j \in \mathcal{D}_{\mathcal{P}}$ are in the same connected component, there exists a path containing edges e_{2i-1} and e_{2i} and it does not go through vertices in $\mathcal{S}$. Therefore, by construction of (23) in Lemma I.3, all the mappings $f_i^{\mathcal{P}}$ does not change the value for inputs $\mathsf{Meta}_F(u)$ for all $u \in \mathcal{S}$. Namely, $f^{\mathcal{P}}(u, \mathcal{U}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. Finally, we set $\tilde{f} = (\circ_{\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)} f^{\mathcal{P}}) \circ f$ to be the composition of f and all $f^{\mathcal{P}}$. We have $\tilde{f}(u, \mathcal{T}_u) = (\circ_{\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)} f^{\mathcal{P}})(u, \mathcal{U}_u) = (u, \mathcal{U}_u)$, as desired. Moreover, $\tilde{f}$ is indeed a proper isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E} \triangle (\bigcup_{\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)} \mathcal{D}_{\mathcal{P}})) = \mathsf{twist}(G(F), \tilde{\mathcal{E}})$.

Conversely, suppose $\tilde{\mathcal{E}}$ satisfies that $|\mathcal{P} \cap \tilde{\mathcal{E}}| \not\equiv |\mathcal{P} \cap \mathcal{E}| \pmod 2$ for some $\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)$. We will prove that any proper isomorphism $\tilde{f}$ from $G(X)$ to $\mathsf{twist}(G(X), \tilde{\mathcal{E}})$ cannot satisfy $\tilde{f}(u, \mathcal{T}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. To prove the result, it suffices to prove that any proper isomorphism $\hat{f}$ from $\mathsf{twist}(G(X), \mathcal{E})$ to $\mathsf{twist}(G(X), \tilde{\mathcal{E}})$ cannot satisfy $\hat{f}(u, \mathcal{U}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S}$. Let $\mathcal{V}_F^{\mathcal{P}} := \bigcup_{\{x,y\} \in \mathcal{P}} \{x, y\} \subset \mathcal{V}_F$ be the set of vertices associated to the connected component $\mathcal{P}$, and let $\mathcal{V}_G^{\mathcal{P}} := \bigcup_{x \in \mathcal{V}_F^{\mathcal{P}}} \mathsf{Meta}_F(x)$. It thus suffices to prove that any proper isomorphism $f^{\mathcal{P}} : \mathcal{V}_G^{\mathcal{P}} \to \mathcal{V}_G^{\mathcal{P}}$ from the induced subgraph $G^{\mathcal{P}} := \mathsf{twist}(G(X), \mathcal{E})[\mathcal{V}_G^{\mathcal{P}}]$ to the induced subgraph $H^{\mathcal{P}} := \mathsf{twist}(G(X), \tilde{\mathcal{E}})[\mathcal{V}_G^{\mathcal{P}}]$ cannot satisfy $f^{\mathcal{P}}(u, \mathcal{U}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S} \cap \mathcal{V}_F^{\mathcal{P}}$.

The proof follows the same technique as Lemma I.7. For each $x \in \mathcal{V}_F^{\mathcal{P}} \backslash \mathcal{S}$, pick an arbitrary meta vertex $(x, \mathcal{U}_x) \in \mathsf{Meta}_F(x)$. Combined with all $(u, \mathcal{U}_u)$ for $u \in \mathcal{S} \cap \mathcal{V}_F^{\mathcal{P}}$, now each vertex $x \in \mathcal{V}_F^{\mathcal{P}}$ is associated with a set $\mathcal{U}_x$. First consider the base case when $f^{\mathcal{P}}(x, \mathcal{U}_x) = (x, \mathcal{U}_x)$ for all $x \in \mathcal{V}_F^{\mathcal{P}}$. It can be proved that the set $\{\{(x, \mathcal{U}_x), (y, \mathcal{U}_y)\} : \{x, y\} \in \mathcal{P}\}$ contains an odd/even number of edges in $G^{\mathcal{P}}$ but an even/odd number of edges in $H^{\mathcal{P}}$, i.e., their parity differs. This is because $H^{\mathcal{P}}$ can be obtained from $G^{\mathcal{P}}$ by twisting the edge set $(\mathcal{E} \triangle \tilde{\mathcal{E}}) \cap \mathcal{P}$, which contains an *odd* number of edges. Therefore, $f^{\mathcal{P}}$ is not a proper isomorphism from $G^{\mathcal{P}}$ to $H^{\mathcal{P}}$ in this case. For the induction step, consider gradually changing the output $f^{\mathcal{P}}(w, \mathcal{U}_w) = (w, \mathcal{U}_w)$ to $f^{\mathcal{P}}(w, \mathcal{U}_w) = (w, \tilde{\mathcal{U}}_w)$ for each $w \in \mathcal{V}_F^{\mathcal{P}} \backslash \mathcal{S}$ where $(w, \tilde{\mathcal{U}}_w) \in \mathsf{Meta}_F(w)$ can be an arbitrary meta vertex. It can be proved that the parity defined above does not change throughout the process (following the proof of Lemma I.7). We have thus proved the induction step, and in all cases there does not exist a proper isomorphism $f^{\mathcal{P}} : \mathcal{V}_G^{\mathcal{P}} \to \mathcal{V}_G^{\mathcal{P}}$ from $G^{\mathcal{P}}$ to $H^{\mathcal{P}}$ satisfying $f^{\mathcal{P}}(u, \mathcal{U}_u) = (u, \mathcal{U}_u)$ for all $u \in \mathcal{S} \cap \mathcal{V}_F^{\mathcal{P}}$. □

Theorem I.9 partially answers the question of how to construct a proper isomorphism f from a Fürer graph $G(F)$ to another Fürer graph $\mathsf{twist}(G(F), \mathcal{E})$ when the mapped outputs $f(\xi)$ are specified for several given inputs $\xi \in \mathcal{V}_G$. However, it does not fully address the problem, because it does not consider the case when two or more inputs ξ are from the same set $\mathsf{Meta}_F(u)$ for some $u \in \mathcal{V}_F$. In the following, we will focus on this general setting. We first consider the special case when all ξ are from the same meta vertex set $\mathsf{Meta}_F(u)$. We have the following result:

Lemma I.10. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be either the original or twisted Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$. Let $u \in \mathcal{V}_F$ and $(u, \mathcal{T}^1), (u, \mathcal{U}^1), \cdots, (u, \mathcal{T}^k), (u, \mathcal{U}^k) \in \mathsf{Meta}_F(u)$ be vertices in $\mathcal{V}_G$. Then, there exists a set $\mathcal{D} \subset \mathcal{N}_F(u)$ such that $\mathcal{T}^i \triangle \mathcal{U}^i = \mathcal{D}$ for all $i \in [k]$, if and only if there exists a proper isomorphism f from graph $G(F)$ to graph $\mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E} \subset \mathcal{E}_F$ with $|\mathcal{E}| \bmod 2 = 0$, such that $f(u, \mathcal{T}^i) = (u, \mathcal{U}^i)$ for all $i \in [k]$.*

Proof. "$\Rightarrow$". Let $\mathcal{D} \subset \mathcal{N}_F(u)$ satisfy that $\mathcal{T}^i \triangle \mathcal{U}^i = \mathcal{D}$ for all $i \in [k]$. Clearly, $|\mathcal{D}| \bmod 2 = 0$. Denote $\mathcal{D} = \{v_1, \cdots, v_{2l}\}$. Similar to the construction in (24), construct a mapping $f : \mathcal{V}_G \to \mathcal{V}_G$ to be $f = f_l \circ \cdots \circ f_1$, where

$$f_j(w, \mathcal{W}) = \begin{cases} (w, \mathcal{W} \triangle \{v_{2j-1}, v_{2j}\}) & \text{if } w = u, \\ (w, \mathcal{W}) & \text{otherwise.} \end{cases}$$

Using a similar analysis, we obtain that f is a proper isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \bigcup_{j=1}^{2l} \{\{u, v_j\}\})$ and $f(u, \mathcal{T}^i) = (u, \mathcal{T}^i \triangle \mathcal{D}) = (u, \mathcal{U}^i)$ for all $i \in [k]$.

"$\Leftarrow$". Assume there does not exist $\mathcal{D} \subset \mathcal{N}_F(u)$ satisfying $\mathcal{T}^i \triangle \mathcal{U}^i = \mathcal{D}$ for all $i \in [k]$. Then, there must exist two indices i, j and a vertex $v \in \mathcal{N}_F(u)$ such that $v \in \mathcal{T}^i \triangle \mathcal{U}^i$ but $v \notin \mathcal{T}^j \triangle \mathcal{U}^j$. We show any proper isomorphism f from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E})$ cannot satisfy both $f(u, \mathcal{T}^i) = (u, \mathcal{U}^i)$ and $f(u, \mathcal{T}^j) = (u, \mathcal{U}^j)$. Let $f(v, \emptyset) = (v, \mathcal{V})$. This is simply due to the following fact:

- If $f(u, \mathcal{T}^i) = (u, \mathcal{U}^i)$, then by definition of isomorphism we have $u \in \emptyset \leftrightarrow v \in \mathcal{T}^i \iff u \in \mathcal{V} \leftrightarrow v \in \mathcal{U}^i$. Since $v \in \mathcal{T}^i \triangle \mathcal{U}^i$, we obtain $u \in \mathcal{V}$;
- If $f(u, \mathcal{T}^j) = (u, \mathcal{U}^j)$, then by definition of isomorphism we have $u \in \emptyset \leftrightarrow v \in \mathcal{T}^j \iff u \in \mathcal{V} \leftrightarrow v \in \mathcal{U}^j$. Since $v \notin \mathcal{T}^j \triangle \mathcal{U}^j$, we obtain $u \notin \mathcal{V}$.

This yields a contradiction and concludes the proof. □

We finally consider the most general setting. Our result is present as follows:

Corollary I.11. *Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be either the original or twisted Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$. Let $\{(u_i, \mathcal{T}_i)\}_{i=1}^k \subset \mathcal{V}_G$ and $\{(u_i, \mathcal{U}_i)\}_{i=1}^k \subset \mathcal{V}_G$ be two vertex sets of $G(F)$. Define $\mathcal{R}(u) := \{(\mathcal{T}_i, \mathcal{U}_i) : u_i = u\}$ and let $\mathcal{S} = \{u_i : i \in [k]\}$. The following two items are equivalent:*

- *There exists a proper isomorphism f from graph $G(F)$ to graph $\mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E} \subset \mathcal{E}_F$ with $|\mathcal{E}| \bmod 2 = 0$, such that $f(u_i, \mathcal{T}_i) = (u_i, \mathcal{U}_i)$ for all $i \in [k]$.*
- *For all $v \in \mathcal{V}_F$, there exists $\mathcal{D}_v \subset \mathcal{N}_F(v)$ such that $\mathcal{T}_i \triangle \mathcal{U}_i = \mathcal{D}_v$ holds for all $(\mathcal{T}_i, \mathcal{U}_i) \in \mathcal{R}(v)$.*

Moreover, if the first item holds, then for any $\tilde{\mathcal{E}} \subset \mathcal{E}_F$, there exists a proper isomorphism $\tilde{f}$ from $G(F)$ to $\mathsf{twist}(G(F), \tilde{\mathcal{E}})$ such that $f(u_i, \mathcal{T}_i) = (u_i, \mathcal{U}_i)$ for all $i \in [k]$, if and only if $|\mathcal{P} \cap \tilde{\mathcal{E}}| \equiv |\mathcal{P} \cap \mathcal{E}| \pmod 2$ for all $\mathcal{P} \in \mathsf{CC}_{\mathcal{S}}(F)$.

Proof. First assume the second item does not hold for some $v \in \mathcal{V}_F$. Then by Lemma I.10, there does not exist a proper isomorphism f from graph $G(F)$ to some $\mathsf{twist}(G(F), \mathcal{E})$ such that $f(v, \mathcal{T}_i) = (v, \mathcal{U}_i)$ for all $(\mathcal{T}_i, \mathcal{U}_i) \in \mathcal{R}(v)$. Clearly, the first item of Corollary I.11 does not hold either.

Now assume the second item holds. For each $v \in \mathcal{S}$, pick an arbitrary element in $\mathcal{R}(v)$, denoted as $(\mathcal{T}^v, \mathcal{U}^v)$. We can then invoke Theorem I.9 with $\{\mathcal{T}^v\}_{v \in \mathcal{S}}$ and $\{\mathcal{U}^v\}_{v \in \mathcal{S}}$. Denote f as the proper isomorphism from $G(F)$ to $\mathsf{twist}(G(F), \mathcal{E})$ for some $\mathcal{E}$ with $|\mathcal{E}| \bmod 2 = 0$ returned by Theorem I.9, such that $f(v, \mathcal{T}^v) = (v, \mathcal{U}^v)$ for all $v \in \mathcal{S}$. It remains to prove that for all other elements $(\mathcal{T}_i, \mathcal{U}_i) \notin \{(\mathcal{T}^v, \mathcal{U}^v) : v \in \mathcal{S}\}$, we still have $f(u_i, \mathcal{T}_i) = (u_i, \mathcal{U}_i)$.

Observe that the construction of f in Theorem I.9 has the form $f(w, \mathcal{W}) = (w, \mathcal{W} \triangle \mathcal{D}_w)$ for all $(w, \mathcal{W}) \in \mathcal{V}_G$ where $\mathcal{D}_w$ is a fixed set for each $w \in \mathcal{V}_F$ (which can be seen from the proof of Theorem I.9). Under the notation above, we have $\mathcal{T}^v \triangle \mathcal{D}_v = \mathcal{U}^v$ for all $v \in \mathcal{S}$. If $f(u_i, \mathcal{T}_i) \neq (u_i, \mathcal{U}_i)$ for some i, then $\mathcal{T}_i \triangle \mathcal{W}_{u_i} \neq \mathcal{U}_i$. This implies that $\mathcal{T}^{u_i} \triangle \mathcal{U}^{u_i} \neq \mathcal{T}_i \triangle \mathcal{U}_i$, which contradicts the second item of Corollary I.11 and concludes the proof. □

Before closing this subsection, we finally introduce a notion called *proper Fürer graphs*, which will be widely used in subsequent analysis.

Definition I.12 (Proper Fürer graphs)**.** A Fürer graph $G(F)$ is called proper, if the base graph $F = (\mathcal{V}_F, \mathcal{E}_F)$ has the following properties:

- F is a connected graph and the degree of any vertex $u \in \mathcal{V}_F$ is at least two;
- There is at least one vertex $u \in \mathcal{V}_F$ with a degree of at least three.

Proposition I.13. *Let $G(F)$ and $H(F)$ be any proper Fürer graph and its twisted graph, respectively. Then both $G(F)$ and $H(F)$ are connected, and the degree of any vertex in both $G(F)$ and $H(F)$ is at least two.*

Proof. By definition of (twisted) Fürer graphs, the degree of a vertex $(u,\mathcal{U})$ in $G(F)$ is $\sum_{v\in\mathcal{N}_F(u)} |\{(v,\mathcal{V}) \in \mathsf{Meta}_F(v) : u \in \mathcal{V} \leftrightarrow v \in \mathcal{U}\}|$. By the assumption that $|\mathcal{N}_F(v)| \geq 2$ for all $v \in \mathcal{V}_F$, the degree of a vertex $(u,\mathcal{U})$ is always $\sum_{v\in\mathcal{N}_F(u)} 2^{|\mathcal{N}_F(v)|-2} \geq \sum_{v\in\mathcal{N}_F(u)} 1 \geq 2$. The case of $H(F)$ is similar. Thus all vertices in both $G(F)$ and $H(F)$ have a degree of at least two.

We next investigate the connectivity of $G(F)$ and $H(F)$. Denote u as any vertex in $G(F)$ or $H(F)$ with a degree of at least 3. We first show that any two vertices $(u,\mathcal{T}), (u,\mathcal{U}) \in \mathcal{V}_G$ satisfying $|\mathcal{T}\triangle\mathcal{U}| = 2$ are in the same connected component. To see this, pick any $w \in \mathcal{N}_F(u)\backslash(\mathcal{T}\triangle\mathcal{U})$ (which exists since $|\mathcal{N}_F(u)| \geq 3$), and consider the two vertices $(w,\emptyset)$ and $(w,\mathcal{W})$ satisfying $u \in \mathcal{W}$ (the existence of $\mathcal{W}$ is due to $w \in \mathcal{N}_F(u)$ and $|\mathcal{N}_F(w)| \geq 2$). Then,

- If $\{(w,\emptyset),(u,\mathcal{T})\}$ is an edge of $G(F)/H(F)$, then $\{(w,\emptyset),(u,\mathcal{U})\}$ is an edge of $G(F)/H(F)$ (because $w \in \mathcal{T}$ if and only if $w \in \mathcal{U}$);
- If $\{(w,\emptyset),(u,\mathcal{T})\}$ is not an edge of $G(F)/H(F)$, then $\{(w,\mathcal{W}),(u,\mathcal{T})\}$ is an edge of $G(F)/H(F)$ (because $u \notin \emptyset$ but $u \in \mathcal{W}$). Therefore, $\{(w,\mathcal{W}),(u,\mathcal{U})\}$ is an edge of $G(F)/H(F)$ (because $w \in \mathcal{T}$ if and only if $w \in \mathcal{U}$).

In both cases, there is a path from $(u,\mathcal{T})$ to $(u,\mathcal{U})$. Next, we can simply remove the assumption $|\mathcal{T}\triangle\mathcal{U}| = 2$: any $(u,\mathcal{T}),(u,\mathcal{U}) \in \mathcal{V}_G$ are also in the same connected component. Finally, for any vertex $(x,\mathcal{X})$ in graph $G(F)/H(F)$, there is a path from $(x,\mathcal{X})$ to some vertex in $\mathsf{Meta}_F(u)$ since F is connected. Using $\mathsf{Meta}_F(u)$ as a "transit set", we have proved that the graph $G(F)/H(F)$ is connected. □

I.2. Simplified pebbling games for Fürer graphs

In Section 6, we developed a unified analyzing framework for all types of WL algorithms based on pebbling games. While the game viewpoint provides interesting and novel insights into the power of different algorithms, it is still quite challenging to directly applying such games for Fürer graphs due to their sophisticated structure. In this subsection, we propose a class of simplified pebbling game motivated from Fürer (2001), which makes our analysis much easier.

We begin by introducing the augmented Fürer graphs defined as follows:

Definition I.14 (Augmented Fürer graphs)**.** Let $G(F) = (\mathcal{V}_G, \mathcal{E}_G)$ be either the original or twisted *proper* Fürer graph of $F = (\mathcal{V}_F, \mathcal{E}_F)$ where $\mathcal{V}_F = [n]$. Let $\tilde{G}(F)$ be the graph augmented from $G(F)$ in the following way: for each $u \in \mathcal{V}_F$, add a chain C_u of length $u+1$ and link one endpoint of C_u to all vertices in $\mathsf{Meta}_F(u)$ (see Figure 3 for an illustration). The vertices on the chains are called *auxiliary vertices*. For each vertex ξ in $\tilde{G}(F)$, by construction it is associated with a vertex u in the base graph. We call u is the *base vertex* of ξ and denote $B(\xi) = u$.

The main motivation of Definition I.14 is that the added auxiliary vertices help distinguish the sets $\mathsf{Meta}_F(u)$ for different u since the lengths of chains C_u are different. Indeed, let A be any SWL algorithm containing a local aggregation or any FWL-type algorithm, and consider playing the pebbling game for A on augmented Fürer graphs $\tilde{G}(F)$ and its twisted version $\tilde{H}(F)$. We have the following result, which shows that Duplicator's best strategy is to match the base vertices for each pair of pebbles.

Lemma I.15. *Consider the pebbling game for any WL algorithm A played on graphs $\tilde{G}(F)$ and $\tilde{H}(F)$. Let (ξ_G, ξ_H) be the position of any pebbles u/v after any round. If $B(\xi_G) \neq B(\xi_H)$, then Spoiler can win the remaining game.*

Proof. It suffices to consider the (weakest) vanilla SWL algorithm $\mathsf{SWL}(\mathsf{VS})$, because Spoiler has more choices to win when considering more powerful WL algorithms. Also, if $B(\xi_G) \neq B(\xi_H)$ holds for pebbles u, Spoiler can just move pebble v in $\tilde{G}(F)$ in subsequent rounds so that the position of v eventually coincides with pebble u. This is feasible because

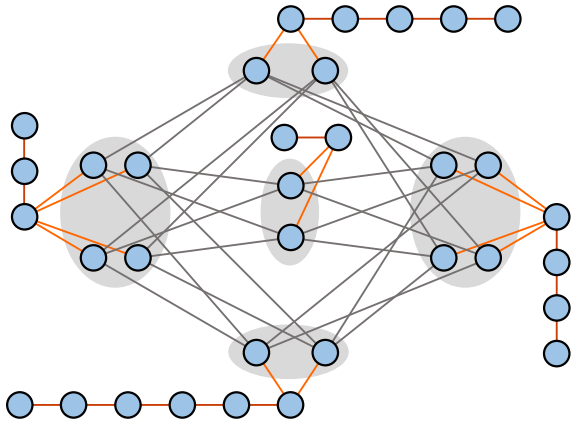


Figure 3. Illustration of the augmented Fürer graph for the Fürer graph in Figure 2. Here, the nodes in gray regions are the vertices of the original Fürer graph and other vertices are from the chains. We also use different colors to distinguish different types of edges.

$\tilde{G}(F)$ is connected (Proposition I.13). Now if the position of pebble v in $\tilde{H}(F)$ does not coincide with pebble u, Spoiler already wins. Therefore, in the remaining proof we can assume that $B(\xi_G) \neq B(\xi_H)$ holds for pebbles v.

Without loss of generality, assume $B(\xi_G) := v_G < v_H := B(\xi_H)$ (note that $\mathcal{V}_F$ is a number set). Spoiler's strategy is then to move pebble v in $\tilde{G}(F)$ towards the endpoint of the chain C_{v_G}. Throughout the process, Duplicator has to keep the pebble v in $\tilde{H}(F)$ located on the chain C_{v_H} (otherwise, the vertices not from the chains must have a degree of at least three (Proposition I.13), and when the degrees of v_G and v_H do not match, Spoiler can win in the next round). When Spoiler finally places pebble v in $\tilde{G}(F)$ to the endpoint of chain C_{v_G}, Duplicator cannot place the other pebble v in $\tilde{H}(F)$ to a vertex of degree 1, so Spoiler can win in the next round. □

Similarly, Duplicator has to ensure that for any pebbles u/v on the two graphs, either they are both placed on the auxiliary vertices, or neither of them is placed on the auxiliary vertices. When they are both placed on the auxiliary vertices, the distance to the corresponding chain endpoint must be the same. It is easy to see that Duplicator can always achieve this goal. Therefore, when Duplicator follows her best strategy, there is no reason for Spoiler to place pebbles on these auxiliary vertices.

We next consider the case when the positions of both pebbles u, v in $\tilde{G}(F)$ correspond to the same base vertex. We have the following result:

Lemma I.16. *Consider the pebbling game for any WL algorithm* A *played on graphs* $\tilde{G}(F)$ *and* $\tilde{H}(F)$*. Let* $u = (\xi_G, \xi_H)$ *and* $v = (\eta_G, \eta_H)$ *be the positions of pebbles* u *and* v *after any round. Assume all pebbles are not placed on auxiliary vertices and correspond to the same base vertex* $w \in \mathcal{V}_F$*. Denote* $\xi_G = (w, \mathcal{T}_G)$*,* $\xi_H = (w, \mathcal{T}_H)$*,* $\eta_G = (w, \mathcal{U}_G)$*,* $\eta_H = (w, \mathcal{U}_H)$*. If* $\mathcal{T}_G \triangle \mathcal{U}_G \neq \mathcal{T}_H \triangle \mathcal{U}_H$*, Spoiler can win the remaining game.*

Proof. The reason why Spoiler can win is essentially given in the proof of Lemma I.10. Similar to the proof of Lemma I.15, we only consider the (weakest) vanilla SWL algorithm $\mathsf{SWL(VS)}$. Since $\mathcal{T}_G \triangle \mathcal{U}_G \neq \mathcal{T}_H \triangle \mathcal{U}_H$, there is a vertex $x \in \mathcal{T}_G \triangle \mathcal{T}_H \triangle \mathcal{U}_G \triangle \mathcal{U}_H$. Clearly, $x \in \mathcal{N}_F(w)$. Spoiler's strategy is then to move pebble v to any of its neighbors $(x, \tilde{\mathcal{U}}_G)$ for some $\tilde{\mathcal{U}}_G$, and Duplicator should move the other pebble v to any of its neighbors $(x, \tilde{\mathcal{U}}_H)$ for some $\tilde{\mathcal{U}}_H$. By definition of Fürer graphs, since $\{(w, \mathcal{U}_G), (x, \tilde{\mathcal{U}}_G)\}$ is an edge of $\tilde{G}(F)$ and $\{(w, \mathcal{U}_H), (x, \tilde{\mathcal{U}}_H)\}$ is an edge of $\tilde{H}(F)$, we have

$$
\begin{aligned}
& (x \in \mathcal{U}_G \leftrightarrow w \in \tilde{\mathcal{U}}_G) = (x \in \mathcal{U}_H \leftrightarrow w \in \tilde{\mathcal{U}}_H) \\
\iff & x \in \mathcal{U}_G \triangle \mathcal{U}_H \leftrightarrow w \in \tilde{\mathcal{U}}_G \triangle \tilde{\mathcal{U}}_H \\
\iff & x \notin \mathcal{T}_G \triangle \mathcal{T}_H \leftrightarrow w \in \tilde{\mathcal{U}}_G \triangle \tilde{\mathcal{U}}_H \\
\iff & (x \in \mathcal{T}_G \leftrightarrow w \in \tilde{\mathcal{U}}_G) \neq (x \in \mathcal{T}_H \leftrightarrow w \in \tilde{\mathcal{U}}_H)
\end{aligned}
$$

Therefore, the isomorphism type of the two vertex pairs $(\xi_G, \tilde{\eta}_G)$ and $(\xi_H, \tilde{\eta}_H)$ is not the same, where $\tilde{\eta}_G := (x, \tilde{\mathcal{U}}_G)$ and $\tilde{\eta}_H := (x, \tilde{\mathcal{U}}_H)$. Spoiler thus wins the game. □

Lemma I.16 further limits Duplicator's strategy when two pebbles share the same base vertex. Moreover, when Duplicator follows her best strategy, it also implies that Spoiler cannot gain extra advantage when he places *multiple* pebbles to positions that belong to the same base vertex. Actually, we will show below that the strategy for both Spoiler and Duplicator can be reduced to focusing only on the information of the base vertices placed by pebbles u, v. In particular, this results in a simplified pebbling game defined as follows.

Simplified pebbling game for augmented Fürer graphs. Let $F = (\mathcal{V}_F, \mathcal{E}_F)$ be the base graph of a proper Fürer graph. The simplified pebbling game is played on F. There are three pebbles u, v, w of different types. Initially, all three pebbles are left outside the graph F. We first describe the game rule for Spoiler, which is similar to Section 6 but is much simpler.

First consider SWL algorithms $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$. If $\mathsf{Pool} = \mathsf{VS}$, Spoiler first places pebble u on any vertex of F and then places pebble v on any vertex of F. If $\mathsf{Pool} = \mathsf{SV}$, Spoiler first places pebble v and then places pebble u.

The game then cyclically executes the following process. Depending on the aggregation scheme $\mathcal{A}$, Spoiler can freely choose one of the following ways to play:

- Local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}} \in \mathcal{A}$. Spoiler first places pebble w adjacent to the vertex placed by pebble v, then swaps pebbles v and w, and finally places pebble w outside the graph F.
- Global aggregation $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}} \in \mathcal{A}$. Spoiler first places pebble w on any vertex of F, then swaps pebbles v and w, and finally places pebble w outside F.
- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}} \in \mathcal{A}$. Spoiler first places pebble w to the position of pebble u, then swaps pebbles v and w, and finally places pebble w outside F.
- Single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vu}} \in \mathcal{A}$. Spoiler swaps the position of pebbles u and v.

The cases of $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$ are similar (symmetric) to $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$, $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$, so we omit them for clarity.

Next consider FWL-type algorithms. Initially, Spoiler simultaneously places pebbles u and v on two vertices of F. The game then cyclically executes the following process. For $\mathsf{LFWL}(2)$, Spoiler first places pebble w to some vertex in $\mathcal{N}^1_F(v)$, then either swaps pebbles v, w or swaps pebbles u, w, and finally places w outside the graph F. The cases of $\mathsf{SLFWL}(2)$ and $\mathsf{FWL}(2)$ are similar, expect that $\mathcal{N}^1_F(v)$ is replaced by $\mathcal{N}^1_F(u) \cup \mathcal{N}^1_F(v)$ and $\mathcal{V}_F$, respectively.

We next describe the game rule for Duplicator, which is of a very different kind. In brief, she maintains a subset $\mathcal{Q}$ of connected components $\mathcal{Q} \subset \mathsf{CC}_{\mathcal{S}}(F)$ (Definition I.8) where the set $\mathcal{S}$ contains vertices of F on which the pebbles u, v, w are currently located. Initially, $\mathcal{Q} := \mathsf{CC}_{\emptyset}(F) = \{\mathcal{E}_F\}$. Note that throughout the game, Spoiler only performs three types of basic operations: (i) add a pebble/two pebbles and place it/them on vertices of F; (ii) remove a pebble and leave it outside the graph F; (iii) swap the positions of two pebbles. Once Spoiler performs an operation above, Duplicator will update $\mathcal{Q}$ according to the following rules so that the parity of $|\mathcal{Q}|$ is always odd throughout the game.

- When Spoiler places some pebble(s) on vertices of F, there are two cases. If $\mathsf{CC}_{\mathcal{S}}(F)$ does not change, then Duplicator does nothing. Otherwise, the presence of new pebbles will split some connected components into a set of smaller regions. For each original connected component $\mathcal{P} \subset \mathcal{E}_F$ that is split into $\mathcal{P}_1, \cdots, \mathcal{P}_k$ with $\bigcup_{i=1}^k \mathcal{P}_i = \mathcal{P}$, Duplicator can replace $\mathcal{Q}$ by $\tilde{\mathcal{Q}} = (\mathcal{Q} \backslash \mathcal{P}) \cup \{\mathcal{P}_{j_1}, \cdots, \mathcal{P}_{j_l}\}$ for some $j_1, \cdots, j_l \in [k]$, such that $|\tilde{\mathcal{Q}}| \bmod 2 = 1$. In other words, Duplicator updates the set $\mathcal{Q}$ by removing the old connected component $\mathcal{P}$ (if in the set) and adding some new partitioned components, while ensuring that the parity of the size of $\mathcal{Q}$ does not change.
- When Spoiler removes a pebble and leave it outside the graph F, there are also two cases. If $\mathsf{CC}_{\mathcal{S}}(F)$ does not change, then Duplicator does nothing. Otherwise, the removal of a pebble will merge several connected components $\mathcal{P}_1, \cdots, \mathcal{P}_k$ into a larger one $\mathcal{P} = \bigcup_{i=1}^k \mathcal{P}_i$. Duplicator then replaces $\mathcal{Q}$ by either $\tilde{\mathcal{Q}} = \mathcal{Q} \backslash \{\mathcal{P}_1, \cdots, \mathcal{P}_k\}$ or $\tilde{\mathcal{Q}} = (\mathcal{Q} \backslash \{\mathcal{P}_1, \cdots, \mathcal{P}_k\}) \cup \mathcal{P}$, depending on which one satisfies $|\tilde{\mathcal{Q}}| \bmod 2 = 1$. In other words, Duplicator updates the set $\mathcal{Q}$ by removing these small connected components and optionally adding the merged component to preserve parity.
- When Spoiler swaps the positions of two pebbles, $\mathsf{CC}_{\mathcal{S}}(F)$ clearly does not change and thus Duplicator does nothing.

For the case of local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, there is an extra constraint for Duplicator: after Spoiler places pebble w adjacent to v and Duplicator updates $\mathcal{Q}$, Duplicator should additionally ensure that $\{\{v, w\}\} \notin \mathcal{Q}$. Similar game rule applies for local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$.

After any round, Spoiler wins if pebble u is adjacent to v and $\{\{u, v\}\} \in \mathcal{Q}$. In other words, Spoiler wins if there is a connected component in $\mathcal{Q}$ with only one edge. Finally, Duplicator wins if Spoiler cannot win after any number of rounds.

Below, we will prove that the simplified pebbling game designed above is actually equivalent to the original pebbling game. Importantly, the simplified pebbling game is played on the base graph F rather than the sophisticated (augmented) Fürer graphs and avoids the complicated vertex selection procedure (Definition 6.1), which greatly eases the analysis of players' strategies.

Theorem I.17. *Let $\tilde{G}(F)$ and $\tilde{H}(F)$ be any augmented proper Fürer graph and its twisted version for some base graph F. For any WL algorithm A considered in this paper, Spoiler can win the corresponding pebbling game on graphs $\tilde{G}(F)$ and $\tilde{H}(F)$ if and only if he can win the simplified pebbling game on graph F.*

Proof. In the original pebbling game, let $\tilde{H}(F) = \mathsf{twist}(\tilde{G}(F), \mathcal{E})$ for some $\mathcal{E}$ with $|\mathcal{E}| \bmod 2 = 1$. Based on Lemma I.15, after any round we can assume that the pebbles u, v are placed on $u = (\xi_G, \xi_H)$, $v = (\eta_G, \eta_H)$ with matching base vertices, i.e., we can denote $\xi_G = (x, \mathcal{T}_x)$, $\xi_H = (x, \mathcal{U}_x)$, $\eta_G = (y, \mathcal{T}_y)$, $\eta_H = (y, \mathcal{U}_y)$. We also assume that the condition of Lemma I.16 holds when $x = y$. The proof is divided into the following parts.

Part 1 (understanding the relationship between the two types of pebbling games). Consider two game states with different pebble positions:

- State 1: the positions of pebbles are $u = ((x, \mathcal{T}_x), (x, \mathcal{U}_x^{(1)}))$, $v = ((y, \mathcal{T}_y), (y, \mathcal{U}_y^{(1)}))$;
- State 2: the positions of pebbles are $u = ((x, \mathcal{T}_x), (x, \mathcal{U}_x^{(2)}))$, $v = ((y, \mathcal{T}_y), (y, \mathcal{U}_y^{(2)}))$.

In other words, the positions of pebbles on graph $\tilde{G}(F)$ are the same for the two states, but the positions of pebbles on graph $\tilde{H}(F)$ differ. By Corollary I.11, there is a proper isomorphism f from $\tilde{H}(F)$ to $\mathsf{twist}(\tilde{H}(F), \tilde{\mathcal{E}})$ for some $\tilde{\mathcal{E}}$ with $|\tilde{\mathcal{E}}| \bmod 2 = 0$, such that $f(x, \mathcal{U}_x^{(1)}) = (x, \mathcal{U}_x^{(2)})$ and $f(y, \mathcal{U}_y^{(1)}) = (y, \mathcal{U}_y^{(2)})$. Note that the second bullet of Corollary I.11 is satisfied since we assume that Duplicator follows the strategy of Lemma I.16 and thus $\mathcal{U}_x^{(1)} \triangle \mathcal{U}_y^{(1)} = \mathcal{T}_x \triangle \mathcal{T}_y = \mathcal{U}_x^{(2)} \triangle \mathcal{U}_y^{(2)}$ when $x = y$. Now using Corollary I.11 again, there is a proper *automorphism* $\tilde{f}$ of $\tilde{H}(F)$ satisfying $\tilde{f}(x, \mathcal{U}_x^{(1)}) = (x, \mathcal{U}_x^{(2)})$ and $\tilde{f}(y, \mathcal{U}_y^{(1)}) = (y, \mathcal{U}_y^{(2)})$, if and only if $|\tilde{\mathcal{E}} \cap \mathcal{P}| \bmod 2 = 0$ for all $\mathcal{P} \in \mathsf{CC}_{\{x,y\}}(F)$. In other words, if $|\tilde{\mathcal{E}} \cap \mathcal{P}| \bmod 2 = 0$ for all $\mathcal{P} \in \mathsf{CC}_{\{x,y\}}(F)$, then the two states are equivalent.

Similarly, since $\tilde{H}(F) = \mathsf{twist}(\tilde{G}(F), \mathcal{E})$, one can also find for each $i = 1, 2$ a proper isomorphism f_i from $\tilde{G}(F)$ to $\mathsf{twist}(\tilde{H}(F), \tilde{\mathcal{E}}_i)$, such that $f(x, \mathcal{T}_x) = (x, \mathcal{U}_x^{(i)})$ and $f(y, \mathcal{T}_y) = (y, \mathcal{U}_y^{(i)})$. Based on the above analysis, whether Spoiler can win the game at state i will thus depend purely on the set

$$\begin{aligned}\mathcal{Q}_i &= \{\mathcal{P} \in \mathsf{CC}_{\{x,y\}}(F) : |\mathcal{E}_i \cap \mathcal{P}| \bmod 2 = 0\} \\ &= \{\mathcal{P} \in \mathsf{CC}_{\{x,y\}}(F) : |(\mathcal{E} \triangle \mathcal{E}_i) \cap \mathcal{P}| \bmod 2 = 1\}.\end{aligned}$$

Namely, if $\mathcal{Q}_1 = \mathcal{Q}_2$, then the two states are equivalent. This is why in the simplified pebbling game Duplicator only maintains the set $\mathcal{Q}$, which has a similar meaning to $\mathcal{Q}_i$.

Part 2 (regarding vertex selection). We show vertex selection in Definition 6.1 can be simplified to satisfy $|\mathcal{S}^{\mathsf{S}}| = |\mathcal{S}^{\mathsf{D}}| = 1$. First, when $\mathcal{S}^{\mathsf{S}}$ contains multiple vertices that correspond to different base vertices in F, Duplicator must respond by matching each base vertex separately and merging them to obtain $\mathcal{S}^{\mathsf{D}}$, otherwise Spoiler can win according to Lemma I.15. When Duplicator follows this strategy, there is no reason for Spoiler to choose multiple base vertices. Next, when $\mathcal{S}^{\mathsf{S}}$ contains multiple vertices that correspond to the same base vertex in F, Duplicator must match each $(x, \mathcal{X}) \in \mathcal{S}^{\mathsf{S}}$ with $(x, \mathcal{X} \triangle \mathcal{D}) \in \mathcal{S}^{\mathsf{D}}$ by selecting a set $\mathcal{D}$ (according to Lemma I.16). In this case, Spoiler still does not gain an additional benefit by selecting $\mathcal{S}^{\mathsf{S}}$ with multiple elements. Moreover, it does not make any difference whether Spoiler chooses to move pebbles on $\tilde{G}(F)$ or on $\tilde{H}(F)$. Therefore, the pebbling game can be simplified so that Spoiler directly moves a pebble in $\tilde{G}(F)$ and Duplicator responds by moving the corresponding pebble in $\tilde{H}(F)$. (Nevertheless, note that the vertex selection procedure is still necessary when dealing with auxiliary vertices as in the proof of Lemma I.15).

Part 3 (equivalence between updating pebble positions and updating $\mathcal{Q}$ for Duplicator). Suppose that in a certain round Spoiler places a pebble w to vertex $(z, \mathcal{T}_z)$ in $\tilde{G}(F)$. If the placement of w does not increase the number of connected components, then no matter how Duplicator responds by replacing the other pebble w to $(z, \mathcal{U}_z)$ in $\tilde{H}(F)$, the game is equivalent due to Part 1 and the set $\mathcal{Q}$ should not change, which coincides with the game rule. If the placement of w increases the number of connected components, then how Duplicator chooses the position of the other pebble w will matter. Suppose Duplicator places the other pebble w on the vertex $(z, \mathcal{U}_z)$, then the value of $\mathcal{T}_z \triangle \mathcal{U}_z$ determines the update of $\mathcal{Q}$ by Corollary I.11. Conversely, each possible game rule for updating $\mathcal{Q}$ also corresponds to at least one feasible position $(z, \mathcal{U}_z)$.

For the local aggregation $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, there is an additional restriction that the pebble w should be adjacent to pebble v. Clearly, the presence of w will make a new connected component $\{\{v, w\}\}$. It is easy to see that $\{\{v, w\}\} \notin \mathcal{Q}$, otherwise pebble w is not adjacent to pebble v in $\tilde{H}(F)$. For localized FWL aggregations, although pebble w should also be placed in the neighborhood of some pebble (e.g., $w \in \mathcal{N}^1_{\tilde{G}(F)}(v)$), we may not add this restriction for Duplicator, because if Duplicator does not obey the game rule, Spoiler can always win after this round by swapping a pair of pebbles (e.g., swapping u and w) such that the isomorphism types of pebbles u and v differ between $\tilde{G}(F)$ and $\tilde{H}(F)$.

Similarly, when Spoiler places a pebble w outside the graph $\tilde{G}(F)$, the connected components may merge, and $\mathcal{Q}$ should be updated accordingly while preserving the parity of its size. This matches the design of the simplified pebbling game. Finally, if Spoiler swaps a pair of pebbles, all connected components remains unchanged, so Duplicator does nothing in the simplified pebbling game. □

I.3. Concrete constructions

In this section, we give concrete constructions to prove all results of Theorem 7.1. We split the proof into a collection of lemmas. All the proofs are based on constructing base graphs F and studying the simplified pebbling game developed in Appendix I.2 on F.

Illustration. For clarity, we illustrate the proof of each lemma with a set of figures (Figures 4 to 11). In each of these figures, the node in orange/green/purple responds to the vertex that holds pebble u/v/w, respectively. We use bold red edges to denote connected components in $\mathcal{Q}$ chosen by Duplicator.

Lemma I.18. *There exist two non-isomorphic graphs such that*

- $\mathsf{SWL}(\mathsf{SV})$ *can distinguish them;*
- $\mathsf{SWL}(\mathsf{VS})$ *cannot distinguish them;*
- $\mathsf{PSWL}(\mathsf{VS})$ *cannot distinguish them.*

Proof. The base graph is constructed in Figure 4. We separately consider each algorithm.

We first analyze the simplified pebbling game for algorithm $\mathsf{SWL}(\mathsf{VS})$. Initially, Spoiler should first place pebble u on some vertex. Due to the symmetry of the graph, there are three cases: vertex 4, vertex 2, and vertex 1. We separately consider each case below:

- If Spoiler places pebble u on vertex 4, then the graph is split into two connected components. By symmetry, without loss of generality suppose Duplicator selects the component at the right of u (Figure 4(a)). Next, Spoiler will place pebble v on some vertex. Clearly, his best strategy is to choose vertex 5 (or equivalently, vertex 6), which can further split the connected component into two parts. Duplicator has to respond by choosing the larger part (Figure 4(b)). In the next round, according to the game rule, Spoiler should place pebble w adjacent to pebble v. He'd better place it on vertex 6. Duplicator can respond appropriately without losing the game (Figure 4(c)). Then Spoiler swaps pebbles v and w and leaves w outside the graph. It can be seen that multiple connected components are then merged into a larger one, yielding Figure 4(d). Now the game state is equivalent to Figure 4(b) by symmetry. It is easy to see that Spoiler can never win the game.
- If Spoiler places pebble u on vertex 2, then the connected component remains unchanged, so Duplicator just does nothing (Figure 4(e)). Next, Spoiler will place vertex v on some vertex, e.g., vertex 3 or vertex 4. Regardless of where he places pebble v, Duplicator's strategy is always to choose the rightmost connected component (see Figure 4(f) and Figure 4(h) for the two cases). First consider the case when pebble v is placed on vertex 3 (Figure 4(f)). In the next round, Spoiler should place pebble w adjacent to pebble v. He'd better place it on vertex 4 to further split the connected component. Duplicator just responds by selecting again the rightmost component as shown in Figure 4(g). Spoiler then swaps pebbles v and w and leaves w outside the graph. It can be seen that the game returns to Figure 4(h)). When Spoiler continues to place pebble w adjacent to pebble v, Duplicator again responds by updating the connected component (Figure 4(i)). However, when Spoiler swaps pebbles v and w and leaves w outside the graph, multiple connected components then merges into a whole, as shown in Figure 4(j). Clearly, Spoiler cannot win the game as well.
- If Spoiler places pebble u on vertex 1, we can similarly prove that Spoiler cannot win the game. Actually, placing pebble u on vertex 1 is clearly not optimal.

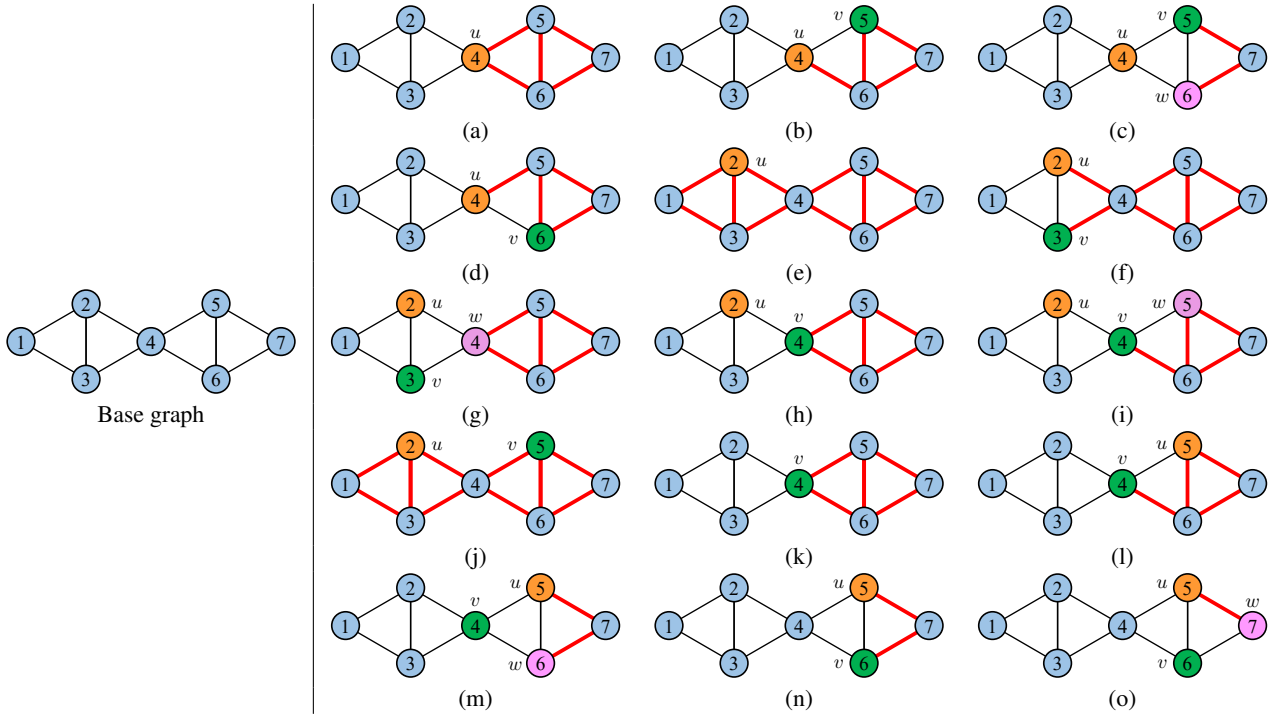


Figure 4. Illustration of the proof of Lemma I.18. When Duplicator follows her optimal strategy, the game process of SWL(VS) corresponds to a sequence of figures, such as (a, b, c, d, ...), (e, f, g, h, i, j, ...), or (e, h, i, j, ...), depending how Spoiler plays. In all cases, Spoiler cannot win. The game process of PSWL(VS) is similar. In contrast, the game process of SWL(SV) corresponds to figures (k, l, m, n, o) and Spoiler eventually wins as shown in figure (o).

We next analyze the simplified pebbling game for algorithm PSWL(VS), which is similar to SWL(VS) except that Spoiler has the additional ability to move pebble u to the position of pebble v. However, when Spoiler performs this operation, the resulting game will simply be equivalent to the three cases studied above, e.g., Figure 4(a) or Figure 4(e), except that pebble v is also present and coincides with u. As already proved above, Spoiler cannot win the game.

We finally analyze the simplified pebbling game for algorithm SWL(SV). In the beginning, Spoiler can first place pebble v on vertex 4, and suppose Duplicator chooses the connected component at the right of v (Figure 4(k)). Spoiler can then place pebble u on vertex 5 to further split this connected component, and Duplicator has to respond by choosing the rightmost component (Figure 4(l)). In the next round, Spoiler can place w on vertex 6. Duplicator has not lost the game yet (see Figure 4(m)). Then it comes to the major difference: when Spoiler swaps pebbles v and w and leaves w outside the graph, the rightmost connected component is *not* merged into a larger one due to the position of pebbles u, v (see Figure 4(n)). Therefore, in the next round, Spoiler can further use pebble w to split the component as shown in Figure 4(o), and Duplicator has no choice other than selecting the connected component $\{\{5, 7\}\}$. Duplicator loses the game after this round. □

Insight into Lemma I.18. The reason why SWL(SV) is stronger lies in the fact that Spoiler can specify the position of pebble u *after seeing Duplicator's response* because pebble v is first placed before pebble u is placed. In this way, Spoiler can exploit such information to better choose the position of pebble u. Importantly, note that pebble u cannot be moved easily according to the game rule, therefore determining its position later may have additional benefits.

Lemma I.19. *There exist two non-isomorphic graphs such that*

- PSWL(VS) *can distinguish them;*
- SWL(VS) *cannot distinguish them;*
- SWL(SV) *cannot distinguish them.*

Proof. The base graph in constructed in Figure 5, which can be seen as a simple adaptation of Figure 4.

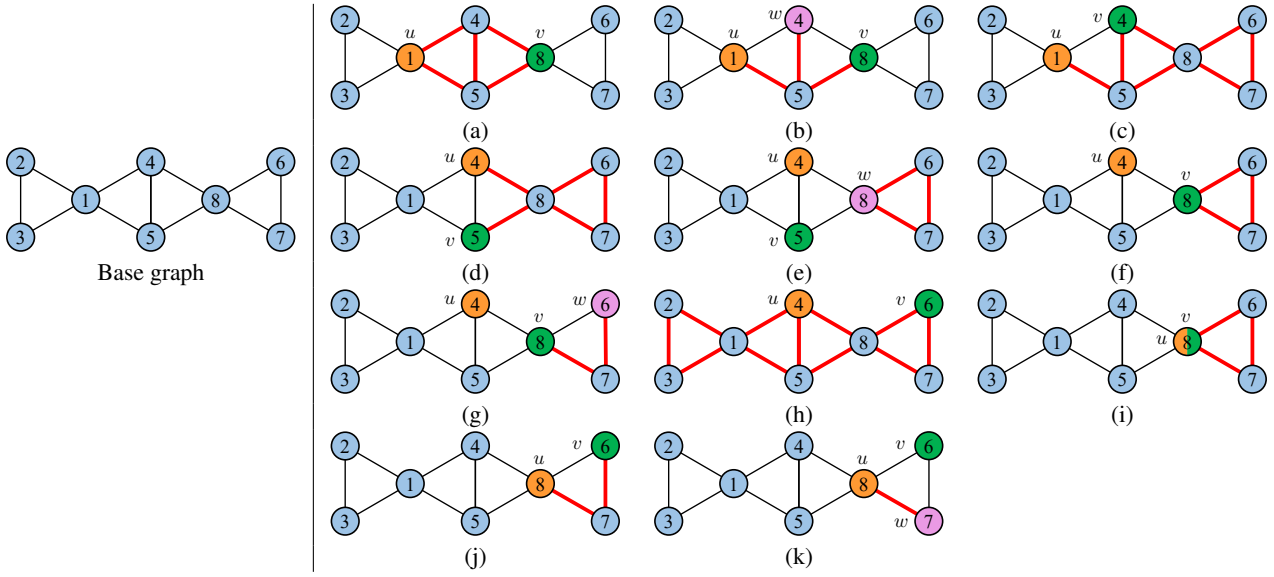


Figure 5. Illustration of the proof of Lemma I.19. When Duplicator follows her optimal strategy, the game process of SWL(VS) (or SWL(SV)) may correspond to figures (a, b, c, ...) or figures (d, e, f, g, h, ...) depending on how Spoiler chooses the initial positions of pebbles u, v. In both cases, Spoiler cannot win. In contrast, the game process of PSWL(VS) corresponds to figures (d, e, f, i, j, k) and Spoiler eventually wins in figure (k).

We first analyze the simplified pebbling game for algorithm SWL(VS) or SWL(SV). Initially, Spoiler should choose the positions for pebbles u, v. We will show that it does not matter whether u or v is placed first. By symmetry, there are mainly two types of strategies which we separately investigate below. Other strategies are similar in analysis and we omit the proof for clarity.

- Strategy 1: Spoiler places pebbles u and v on vertices 1 and 8, respectively. In this case, Duplicator's strategy is to ensure that the middle connected component is selected after pebbles u, v are present, as shown in Figure 5(a). According to the game rule, Spoiler should then place pebble w on some vertex adjacent to v, and clearly, he'd better place w on vertex 4 (or vertex 5 by symmetry). Duplicator can respond appropriately without losing the game (shown in Figure 5(b)). When Spoiler swaps pebbles v, w and leaves w outside the graph, the chosen connected component will be merged (Figure 5(c)). It is easy to see that Spoiler can never win the game after any number of rounds.
- Strategy 2: Spoiler places pebbles u and v on vertices 4 and 5, respectively. By symmetry, suppose Duplicator chooses the connected component on the right (Figure 5(d)). Then Spoiler should place pebble w on some vertex adjacent to v, and he'd clearly place w on vertex 8. Duplicator must respond by choosing the rightmost triangle, resulting in Figure 5(e). When Spoiler swaps pebble v, w and leaves w outside the graph, the triangle component remains unchanged due to the presence of pebble v (Figure 5(f)). In the next round, Spoiler should place pebble w to further split the triangle, like Figure 5(g). However, he cannot win: when he swaps pebble v, w and leaves w outside the graph, all previous components merge into a whole as shown in Figure 5(h). Spoiler has no idea how to win.

We next turn to algorithm PSWL(VS). Initially, the game is the same as SWL(VS) until reaching the state of Figure 5(f). In the next round, Spoiler can resort to the game rule of $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$ and move pebble u to the position of pebble v (Figure 5(i)). Now pebble u becomes useful and Spoiler can easily win the remaining game, like Figure 5(j, k). □

Insight into Lemma I.19. The reason why PSWL(VS) is stronger lies in the fact that Spoiler can change the position of pebble u throughout the game process. In contrast, for SWL(VS) and SWL(SV), pebble u has to be kept fixed once it is placed on the graph, which severely limits the utility of the pebble u in the subsequent game.

Lemma I.20. *There exist two non-isomorphic graphs such that*

- GSWL *can distinguish them;*
- PSWL(SV) *cannot distinguish them.*

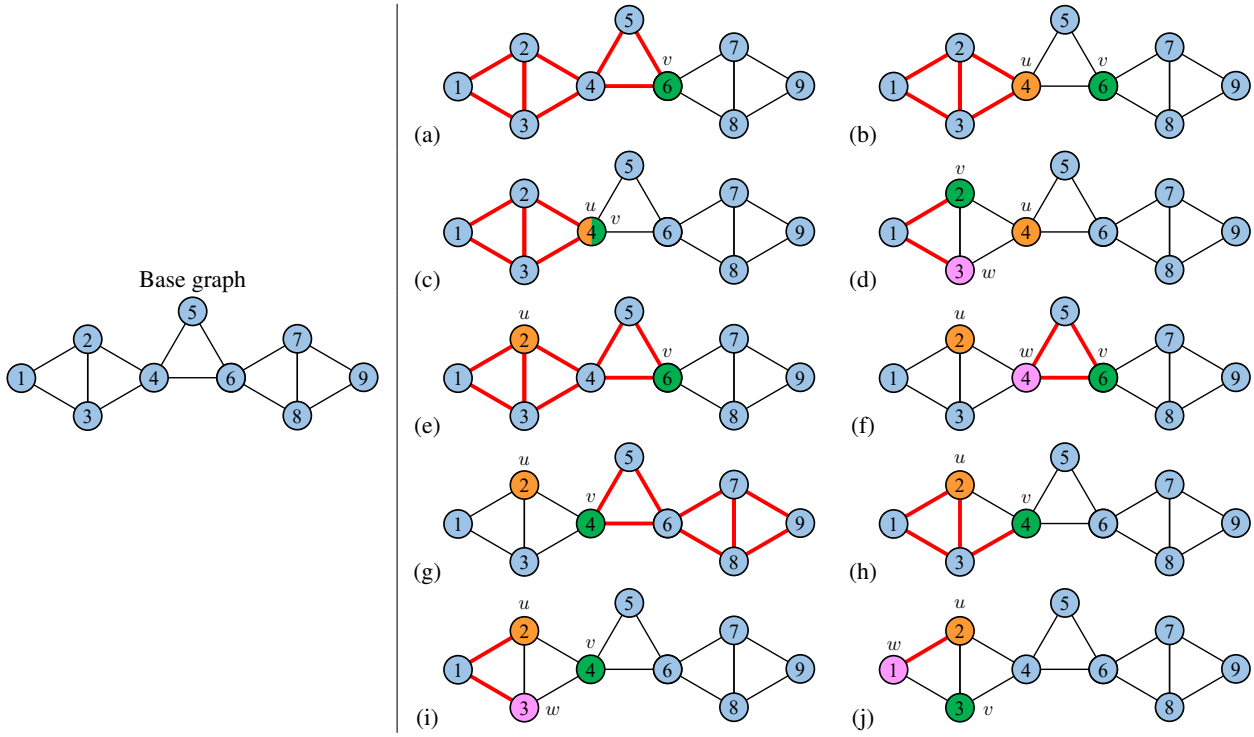


Figure 6. Illustration of the proof of Lemma I.20. When Duplicator follows her optimal strategy, the game process of $\mathsf{PSWL}(\mathsf{SV})$ may correspond to figures (a, b, c, d, ...) or figures (a, e, f, g, ...) depending on how to choose the initial position of pebble u. In both cases, Spoiler cannot win. In contrast, the game process of GSWL corresponds to figures (a, b, c, h, i, j) and Spoiler eventually wins in figure (j).

Proof. The base graph in constructed in Figure 6, which can be seen as a further extension of the counterexample in Figure 4.

We first analyze the simplified pebbling game for algorithm $\mathsf{PSWL}(\mathsf{SV})$. Initially, Spoiler should place pebble v on some vertex. We only consider the case of choosing vertex 6 (or equivalently, vertex 4), which is intuitively the best choice. Other choices can be similarly analyzed and we omit them for clarity. Since the presence of v splits the graph into two connected components, Duplicator should select the larger one (Figure 6(a)). Next, Spoiler should place pebble u on some vertex.

- We first consider the case when Spoiler places u on vertex 4, which further splits the left connected component. In this case, Duplicator just selects the left diamond-shaped component (Figure 6(b)). In the next round, Spoiler will play according to Figure 6(c) by placing pebble w on vertex 4 adjacent to v, swapping v and w, and leaving pebble w outside the graph. Duplicator just does nothing. The remaining game can be illustrated in Figure 6(d), and the analysis is the same as the previous proof of Lemma I.18. In short, Spoiler can never split the red connected component $\{\{1,2\},\{1,3\}\}$ shown in Figure 6(d). Note that although Spoiler can additionally use the game rule of single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, he should better not change the position of u: if he leaves pebble u from vertex 4, the connected component will be merged. Therefore, Spoiler cannot win the game.
- Seeing why Spoiler cannot win in Figure 6(d), let us restart from Figure 6(a) with a different strategy. Suppose this time Spoiler places pebble u on vertex 2 (shown in Figure 6(e)). Since the red connected component remains unchanged, Duplicator does nothing. In the next round, Spoiler should place pebble w on vertex 4 adjacent to v, which splits the red connected component in Figure 6(e) into two parts. However, seeing the position of pebble u, this time Duplicator chooses a different strategy: she selects the upper triangle (Figure 6(f)). When Spoiler swaps pebbles v, w and leaves w outside the graph, the upper triangle is merged into a larger connected component (see Figure 6(g)). It is easy to see that Spoiler still cannot win the game after any number of rounds.

We next turn to algorithm GSWL. Initially, the game is the same as $\mathsf{PSWL}(\mathsf{SV})$ until reaching the state of Figure 5(c). In the next round, Spoiler can choose a different way to play: according to the game rule of $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$, Spoiler can place pebble w on vertex 2 and swap w with u. Clearly, Duplicator has to respond by selecting the left connected component as shown in Figure 6(h). Now the remaining game is easy for Spoiler. As illustrated in Figure 6(i) and Figure 6(j), Spoiler can finally win the game. □

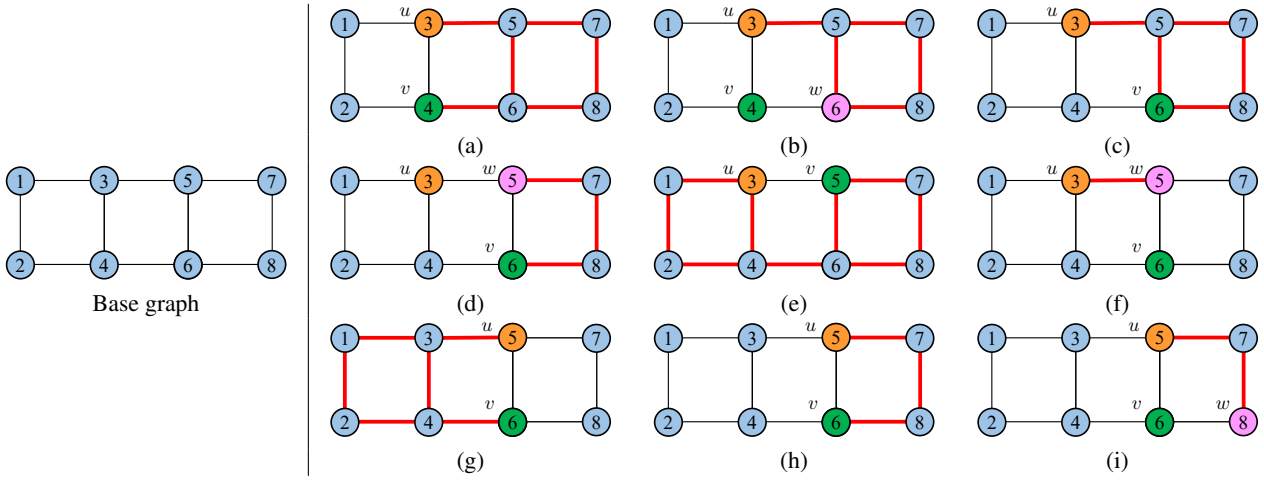


Figure 7. Illustration of the proof of Lemma I.21. When Duplicator follows her optimal strategy, the game process of GSWL may correspond to figures (c, d, e, ...), (c, f, g, ...), (a, b, c, d, e, ...), or (a, b, c, f, g, ...), depending on Spoiler's strategy. In all cases, Spoiler cannot win. In contrast, the game process of SSWL corresponds to figures (a, b, c, d, h, i) and Spoiler eventually wins in figure (i). The game process of LFWL(2) is similar and Spoiler can also win.

Insight into Lemma I.20. The proof of Lemma I.20 clearly shows why global aggregation is more powerful than the corresponding single-point aggregation (Theorem 4.4).

Lemma I.21. *There exist two non-isomorphic graphs such that*

- GSWL *cannot distinguish them;*
- SSWL *can distinguish them;*
- LFWL(2) *can distinguish them.*

Proof. The base graph is constructed in Figure 6, which is precisely the graph originally analyzed in Fürer (2001) and is often called the Fürer grid graph (Qian et al., 2022, Appendix D).

We first analyze the simplified pebbling game for algorithm GSWL. Depending on how Spoiler chooses the initial positions of pebbles u and v, we main consider the two cases illustrated in Figure 7(a) and Figure 7(c) due to symmetry of the graph. Other cases are clearly not optimal. For the first case, Duplicator will select the larger connected component on the right (Figure 7(a)). In the next round, Spoiler may place pebble w on vertex 6 adjacent to pebble v, and Duplicator updates her selected component accordingly (Figure 7(b)). Spoiler then swaps pebbles v and w and leaves w outside the graph, returning to Figure 7(c). What follows is the central part of the proof. Spoiler should clearly place pebble w on vertex 5 to further split the component selected by Duplicator, but he has two different ways to achieve this:

- He plays according to the game rule of $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$. Duplicator knows this information and thus responds by selecting the connected component on the right (see Figure 7(d)). Then Spoiler should swap pebbles w and v and leave w outside the graph. However, this will merge multiple component as shown in Figure 7(e). Clearly, Spoiler cannot win the subsequent game.
- A better choice would be to follow the game rule of $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$ because it can move pebble u to vertex 5 after this round. However, Duplicator knows this information and thus responds differently: she selects the connected component of $\{\{3, 5\}\}$ containing only one edge (see Figure 7(f)). Note that Duplicator does not lose the game, because for global aggregation Duplicator can freely choose a connected component of *one edge* (while for local aggregation she cannot). Now, when Spoiler swaps pebbles w and u and leaves w outside the graph, the component $\{\{3, 5\}\}$ is merged into a larger component, as shown in Figure 7(g), which is equivalent to Figure 7(a) by symmetry. Again, Spoiler cannot win the subsequent game.

We next turn to algorithm SSWL. Initially, the game is the same as GSWL until reaching the state of Figure 7(d). Now it comes to the major difference: Spoiler can play according to the game rule of $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$. This time Duplicator can no longer choose the connected component of $\{\{3, 5\}\}$ since it is prohibited by the game rule. Therefore, her only choice is to select

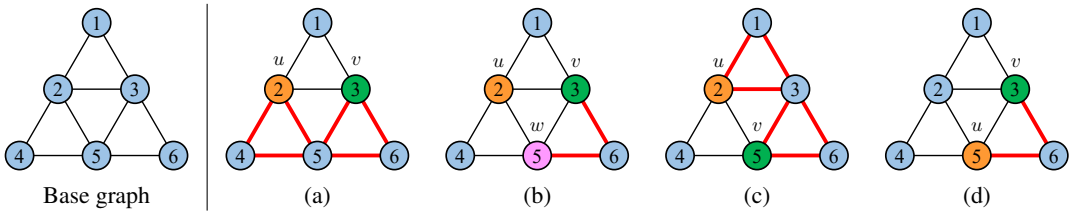


Figure 8. Illustration of the proof of Lemma I.22. When Duplicator follows her optimal strategy, the game process of SSWL may correspond to figures (a, b, c, ...), and Spoiler cannot win. In contrast, the game process of LFWL(2) or SLFWL(2) corresponds to figures (a, b, d) and Spoiler can eventually win.

the rightmost component as shown in Figure 7(d). This then yields Figure 7(h) when Spoiler swaps pebbles u, w and leaves w outside the graph. The remaining game will be quite easy for Spoiler and it is easy to see that Spoiler can win when playing according to Figure 7(i).

We finally turn to algorithm LFWL(2). Initially, the game is also the same as GSWL until reaching the state of Figure 7(d). Now it comes to the major difference: Duplicator does not know whether Spoiler will swap pebbles u, w or swap pebbles v, w. Therefore, depending on Duplicator's response, Spoiler can adopt different strategies:

- If Duplicator chooses the rightmost connected component, then Spoiler swaps pebbles u, w. This corresponds to Figure 7(h) when w is left outside the graph, and we have proved that Spoiler can win.
- If Duplicator chooses the component containing only one edge $\{\{3, 5\}\}$, then Spoiler swaps pebbles v, w. This corresponds to Figure 7(f) and Spoiler already wins after this round.
- Similarly, if Duplicator chooses the component containing only one edge $\{\{5, 6\}\}$, then Spoiler swaps pebbles u, w and wins after this round.

In all cases, Spoiler has a winning strategy. □

Insight into Lemma I.21. The proof of Lemma I.21 shows why local aggregation is more powerful than global aggregation (Theorem 4.4). Importantly, in local aggregation there is an *additional constraint* that Duplicator cannot choose the connected component containing the neighboring edge. The proof also reveals the power of FWL-type algorithms. Intuitively, in FWL-type algorithms Duplicator cannot "see" Spoiler's strategy before making choices, and thus Spoiler can gain an additional advantage by deliberately playing against Duplicator's strategy.

Based on the proof of Lemma I.21, curious readers may ask whether there is an expressivity relationship between SSWL and LFWL(2). However, below we will show that it is not the case: they are actually incomparable (due to Lemmas I.22 and I.23).

Lemma I.22. *There exist two non-isomorphic graphs such that*

- SSWL *cannot distinguish them;*
- SLFWL(2) *can distinguish them;*
- LFWL(2) *can distinguish them.*

Proof. The base graph is constructed in Figure 8.

We first analyze the simplified pebbling game for algorithm SSWL. Initially, Spoiler should place pebbles u and v on vertices of the graph. Due to symmetry, we can assume that Spoiler places u on vertex 2 and places v on vertex 3 (other nonequivalent cases are clearly not optimal). Duplicator then selects the bottom connected component split by pebbles u, v (see Figure 8(a)). In the next round, Spoiler should place w on vertex 5 to further split the connected component. By definition of SSWL, he can play according to the rule of either $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ or $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$. By symmetry, it suffices to analyze the case of $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$. Since Duplicator knows the information that Spoiler plays according to $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$, she selects the connected component in the lower right corner (see Figure 8(b)). Then, Spoiler should swap pebbles v, w and leave w outside the graph, which leads to the merging of multiple connected components. The resulting game, as illustrated in Figure 8(c), is equivalent to Figure 8(a) by symmetry. Therefore, Spoiler cannot win the game.

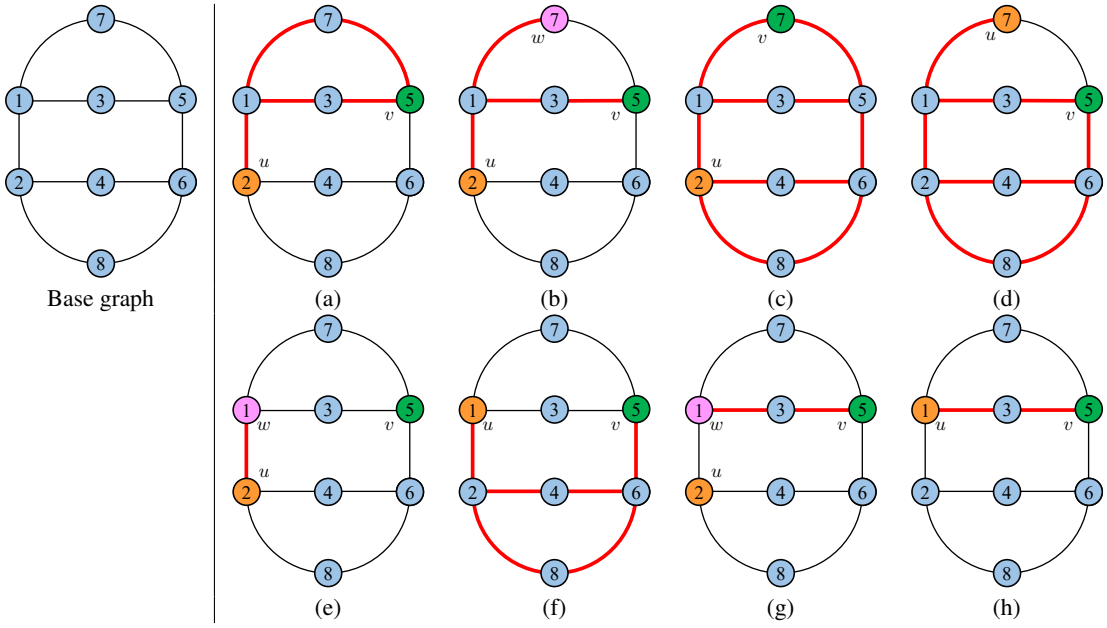


Figure 9. Illustration of the proof of Lemma I.23. When Duplicator follows her optimal strategy, the game process of $\mathsf{LFWL}(2)$ may correspond to figures (a, b, c, ...) or figures (a, b, d, ...) depending on Spoiler's strategy. In both cases, Spoiler cannot win. Similarly, the game process of GSWL may correspond to figures (a, b, c, ...) or figures (a, e, f, ...), and Spoiler still cannot win. In contrast, the game process of SSWL or $\mathsf{SLFWL}(2)$ corresponds to figures (a, g, h) and Spoiler can eventually win.

We next analyze the simplified pebbling game for algorithm $\mathsf{LFWL}(2)$ or $\mathsf{SLFWL}(2)$. Initially, the game is the same as SSWL until reaching the state of Figure 8(b). Now it comes to the major difference: Duplicator does not know whether Spoiler will swap pebbles u, w or swap pebbles v, w. Therefore, Duplicator can only choose either the bottom left component or the bottom right one *at random*, which is equivalent by symmetry. Spoiler can then play against Duplicator's strategy and swap the pebbles so that after leaving pebble w outside the graph, the connected component selected by Duplicator is not merged, as shown in Figure 8(d). Clearly, Spoiler can win the remaining game. □

Insight into Lemma I.22. Lemma I.22 shows the inherent advantage of FWL-type algorithms compared with SWL algorithms, answering an open problem raised in Frasca et al. (2022).

Lemma I.23. *There exist two non-isomorphic graphs such that*

- $\mathsf{LFWL}(2)$ *cannot distinguish them;*
- $\mathsf{SLFWL}(2)$ *can distinguish them;*
- GSWL *cannot distinguish them;*
- SSWL *can distinguish them.*

Proof. The base graph is constructed in Figure 9.

We first analyze the simplified pebbling game for algorithm $\mathsf{LFWL}(2)$. Initially, Spoiler should place pebbles u and v on vertices of the graph. Due to symmetry, there are mainly two cases we need to consider, as shown in Figure 9(a) and Figure 9(f), respectively. Here, we only consider the case of Figure 9(a), where pebble u is placed on vertex 2 and pebble v is placed on vertex 5; the other case is similar to analyze. Duplicator will respond by selecting the top connected component (Figure 9(a)). In the next round, Spoiler should place pebble w adjacent to pebble v. Clearly, he should place it on vertex 3 or 7, which is equivalent. Assume that he places w on vertex 7. Duplicator can easily respond by choosing the larger component (Figure 9(b)). According to the game rule of $\mathsf{LFWL}(2)$, he can either swap pebbles v, w or swap pebbles u, w, and then leaves w outside the graph. As shown in Figure 9(c) and Figure 9(d), in both cases the connected components get merged after leaving pebble w. Clearly, Spoiler cannot win.

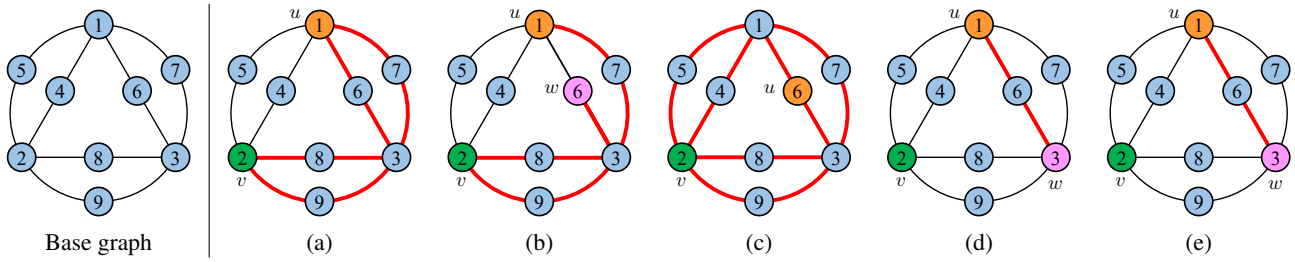


Figure 10. Illustration of the proof of Lemma I.24. When Duplicator follows her optimal strategy, the game process of $\mathsf{SLFWL}(2)$ may correspond to figures (a, b, c, ...), and Spoiler cannot win. In contrast, the game process of $\mathsf{FWL}(2)$ corresponds to figures (a, d, e) and Spoiler can eventually win.

We next turn to algorithm GSWL. Initially, the game is similar (Figure 9(a)). Now Spoiler has the additional choice to place pebble w on vertex 1 according to the game rule of $\mathsf{agg}_{\mathsf{v}}^{\mathsf{G}}$. In this case, Duplicator responds by choosing the connected component $\{\{1,2\}\}$ which contains only one edge (see Figure 9(e)). Note that Duplicator does not lose the game. Then Spoiler will swap pebbles u and w and leaves w outsides the graph, yielding Figure 9(f). Spoiler cannot win the game either.

We finally turn to algorithm SSWL. This time Spoiler can place pebble w adjacent to pebble u according to the game rule of $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$, and Duplicator can only choose either the connected component $\{\{1,3\},\{3,5\}\}$ or $\{\{1,7\},\{7,5\}\}$ (Figure 9(g)). Note that She cannot choose the component $\{\{1,2\}\}$ according to the game rule. After swapping pebbles u, w and leaving w outside the graph (Figure 9(h)), the remaining game is quite easy for Spoiler and he can eventually win. □

Insight into Lemma I.23. Lemma I.23 shows the inherent advantages of "symmetrized" WL algorithms compared with WL algorithms that only aggregate local information of *one* vertex.

Lemma I.24. *There exist two non-isomorphic graphs such that*

- $\mathsf{SLFWL}(2)$ *cannot distinguish them;*
- $\mathsf{FWL}(2)$ *can distinguish them.*

Proof. The base graph in constructed in Figure 10.

We first analyze the simplified pebbling game for algorithm $\mathsf{SLFWL}(2)$. Initially, Spoiler should place pebbles u and v on vertices of the graph. Due to symmetry, we can assume that he places pebble u on vertex 1 and places pebble v on vertex 2. Other nonequivalent choices are clearly not optimal. Duplicator then responds by choosing the largest connect component as shown in Figure 10(a). In the next round, Spoiler can place pebble w on any vertex in $\mathcal{N}_F^1(1) \cup \mathcal{N}_F^1(2)$, namely, any vertex except vertex 3. Due to symmetry, we can assume that he places w on vertex 6. Duplicator can easily respond according to Figure 10(b). No matter how Spoiler swaps pebbles, as long as pebble w is left outside the graph, multiple connected components then merge as shown in Figure 10(c) and Spoiler has no idea how to win.

We next turn to algorithm $\mathsf{FWL}(2)$. Starting from Figure 10(a), this time Spoiler can place pebble w on vertex 3. Then Duplicator should choose an odd number of connected components from the four components: $\{\{1,6\},\{6,3\}\}$, $\{\{1,7\},\{7,3\}\}$, $\{\{2,8\},\{8,3\}\}$, and $\{\{2,9\},\{9,3\}\}$. Regardless of Duplicator's choice, after swapping either pebbles u, w or pebbles v, w, the chosen component are always surrounded by pebbles u and v (Figure 10(e)). Therefore, Spoiler can easily win the remaining game. □

Insight into Lemma I.24. Lemma I.24 shows there is an inherent gap between 2-FWL and all $O(nm)$-complexity algorithms considered in this paper. It can also be used to settle the open problem raised in Frasca et al. (2022).

Lemma I.25. *There exist two non-isomorphic graphs such that*

- $\mathsf{SWL}(\mathsf{SV})$ *can distinguish them;*
- $\mathsf{LFWL}(2)$ *cannot distinguish them.*

Proof. The base graph is constructed in Figure 11.

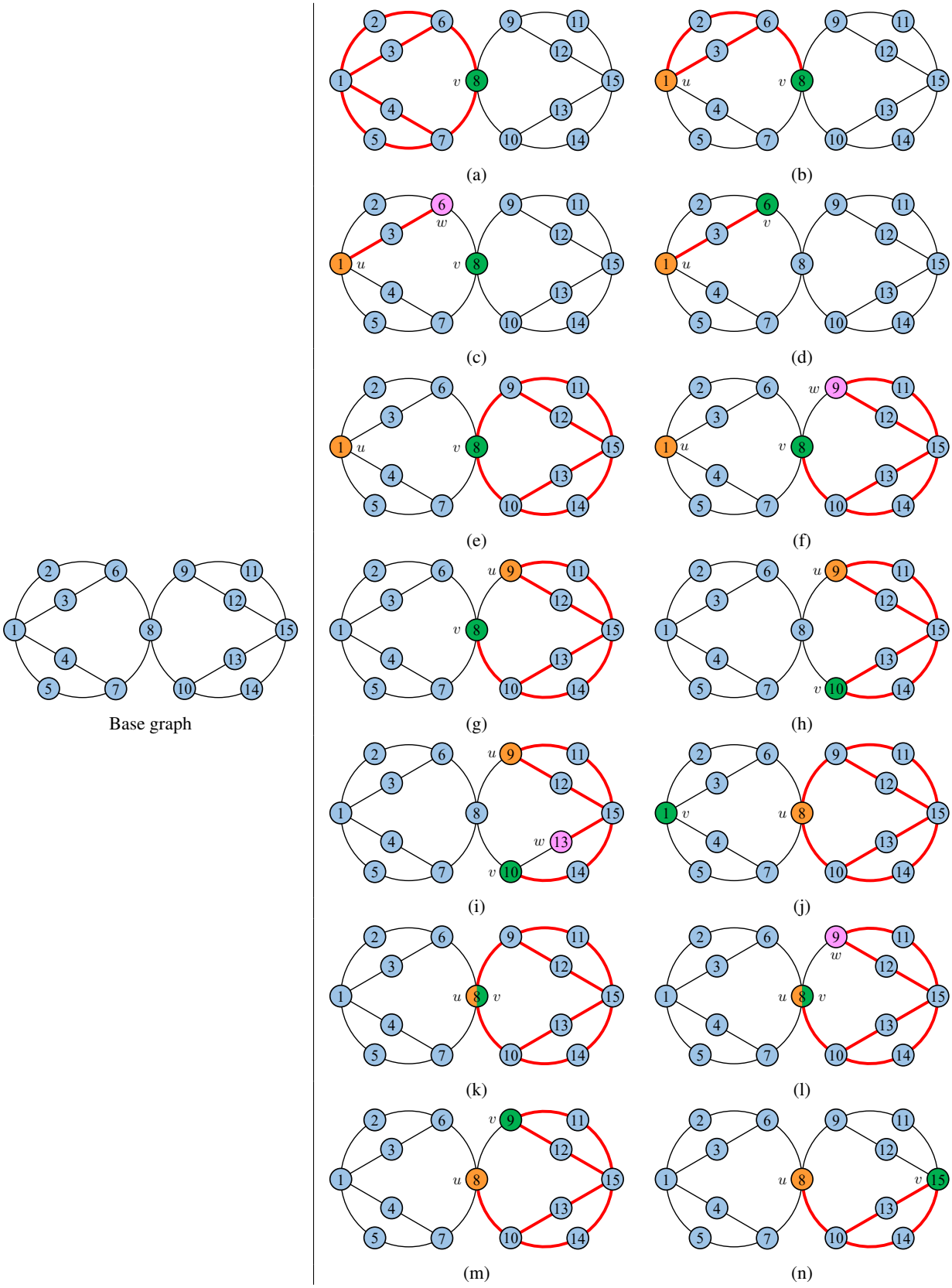


Figure 11. Illustration of the proof of Lemma I.25. When Duplicator follows her optimal strategy, the game process of SWL(SV) corresponds to figures (a, b, c, d), and Spoiler can eventually win the game. In contrast, the game process of LFWL(2) may correspond to figures (e, f, g, h, i, ...) or figures (j, k, l, g, h, i, ...) or figures (j, k, l, m, n, ...), depending how Spoiler swaps pebbles. In both cases, Spoiler cannot win.

We first analyze the simplified pebbling game for algorithm $\mathsf{SWL}(\mathsf{SV})$. Initially, Spoiler places pebble v on vertex 8, which splits the graph into two equal parts. Due to symmetry, suppose Duplicator selects the left component (Figure 11(a)). Spoiler then places pebble u on vertex 1 to further split the connected component. Due to symmetry, suppose Duplicator selects the top-left component (Figure 11(b)). In the next round, Spoiler places pebble w adjacent to v on vertex 6. Duplicator should choose the connected component of either $\{\{1,3\},\{3,6\}\}$ or $\{\{1,2\},\{2,6\}\}$, which is equivalent by symmetry (see Figure 11(c)). Spoiler then swaps pebbles v, w and leaves w outside the graph (Figure 11(d)). The remaining game is straightforward to analyze and Spoiler can easily win.

We next turn to the algorithm $\mathsf{LFWL}(2)$. Initially, Spoiler should simultaneously place pebbles u and v on two vertices of the graph. Without loss of generality, assume that he places one pebble on vertex 8 and places the other pebble on vertex 1 (other cases are similar to analyze). Depending on which pebble is placed on vertex 8, there are two cases:

- Spoiler places pebble u on vertex 1 and places pebble v on vertex 8. In this case, Duplicator responds by choosing the connected component on the right (Figure 11(e)). In the next round, Spoiler should better place pebble w on vertex 9 (or vertex 10) adjacent to pebble v, and Duplicator can respond accordingly (Figure 11(f)). Spoiler should then swap pebbles u and w and leave w outside the graph (Figure 11(g)). In the next round, Spoiler can similarly place pebble w on vertex 10 adjacent to pebble v to further split the connected component. This corresponds to Figure 11(h) after swapping pebbles v and w and leaving w outside the graph. In subsequent rounds, Spoiler can continue to place pebble w adjacent to pebble v (Figure 11(i)). However, whether he swaps pebbles u, w or pebbles v, w, as long as pebble w is left outside the graph, multiple connected components then merge into a whole. Clearly, Spoiler cannot win the game.
- Spoiler places pebble u on vertex 8 and places pebble v on vertex 1. In this case, Duplicator similarly responds by choosing the connected component on the right (Figure 11(j)). In subsequent rounds, Spoiler should gradually move pebble v until it reaches the position of pebble u (Figure 11(k)). Next, Spoiler will continue to place pebble w adjacent to pebble v on vertex 9, and Duplicator can respond accordingly (Figure 11(l)). Spoiler should then either swap pebbles u, w or swap pebbles v, w and leave w outside the graph. The former case corresponds to (Figure 11(g) and has been analyzed. Now consider the latter case, which corresponds to Figure 11(m). In subsequent rounds, Spoiler can perform arbitrary operations, but he can never change the position of pebbles u. Otherwise, the lack of a pebble on vertex 8 will cause the merging of multiple connected components. With the position of pebble u unchanged, the best state Spoiler can achieve is illustrated in Figure 11(i). It is not hard to figure out that Spoiler cannot win the game, either.

In both cases, Spoiler cannot win. □

Insight into Lemma I.25. The reason why $\mathsf{SWL}(\mathsf{SV})$ is stronger in this case is due to the SV pooling strategy. Lemma I.25 shows that $\mathsf{LFWL}(2)$ does not have the ability to implement the SV pooling strategy.

We are now ready to prove Theorem 7.1.

Proof of Theorem 7.1. Theorem 7.1 is a direct consequence of Corollary 4.7, Theorem 5.2, and Lemmas I.18 to I.25. □

J. Proof of Theorems in Appendix A

J.1. Proof of Theorem A.2

We first define several notations. Consider a path $P = (x_0, \cdots, x_d)$ (not necessarily simple) in graph G of length $d \geq 1$. We say P is a *hitting path*, if $x_i \neq x_d$ for all $i \in \{0, 1, \cdots, d-1\}$. Denote $\mathcal{Q}_G^d(u, v)$ to be the set of all hitting paths from node u to node v of length d. Denote $\mathrm{dis}_G^{\mathsf{H}}(u, v)$ as the *hitting time distance* between vertices u and v in graph G, i.e, the average hitting time in a random walk from vertex u to v. Then,

$$\mathrm{dis}_G^{\mathsf{H}}(u, v) = \sum_{d=0}^{\infty} d \cdot \sum_{(x_0, \cdots, x_d) \in \mathcal{Q}_G^d(u,v)} 1 / \left(\prod_{i=0}^{d-1} \deg(x_i) \right).$$

Given a path $P = (x_0, \cdots, x_d)$, define $\omega(P) := (\deg_G(x_1), \cdots, \deg_G(x_{d-1}))$, which is a tuple of length $d-1$. Our proof if based on the following lemma:

Lemma J.1. *Let $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ be two graphs, and let $t \in \mathbb{N}_+$ be a positive integer. Consider any SWL algorithm $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ with $\mathsf{agg}_{\mathsf{u}}^{\mathsf{L}} \in \mathcal{A}$ and denote $\chi^{(t)}$ to be the color mapping after iteration t. Given nodes $u, v \in \mathcal{V}_G$*

and $x, y \in \mathcal{V}_H$*, if* $\chi_G^{(t)}(u,v) = \chi_H^{(t)}(x,y)$*, then* $\{\!\{\omega(Q) : Q \in \mathcal{Q}_G^t(v,u)\}\!\} = \{\!\{\omega(Q) : Q \in \mathcal{Q}_H^t(y,x)\}\!\}$.

Proof. The proof is based on induction over t. For the base case of $t = 1$, it is easy to see that $\{\!\{\omega(Q) : Q \in \mathcal{Q}_G^1(v,u)\}\!\}$ depends only on whether $\{u,v\} \in \mathcal{E}_G$ or not. Clearly, if $\chi_G^{(1)}(u,v) = \chi_H^{(1)}(x,y)$, then $\{u,v\} \in \mathcal{E}_G \leftrightarrow \{x,y\} \in \mathcal{E}_G$ holds (Lemma E.4), implying $\{\!\{\omega(Q) : Q \in \mathcal{Q}_G^1(v,u)\}\!\} = \{\!\{\omega(Q) : Q \in \mathcal{Q}_H^1(y,x)\}\!\}$.

Now assume that the lemma holds for all $t \leq T$, and consider the case of $t = T+1$. When $\chi_G^{(T+1)}(u,v) = \chi_H^{(T+1)}(x,y)$, by definition of $\mathsf{agg}_\mathsf{u}^\mathsf{L}$ we have

$$\{\!\{\chi_G^{(T)}(u,w) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{\chi_H^{(T)}(x,z) : z \in \mathcal{N}_H(y)\}\!\}.$$

Since $T \geq 1$, we have $\chi_G^{(T)}(u,w) = \chi_H^{(T)}(x,z) \implies \deg_G(w) = \deg_H(z)$ for any $w \in \mathcal{V}_G$ and $z \in \mathcal{V}_H$. Therefore,

$$\{\!\{(\chi_G^{(T)}(u,w), \deg_G(w)) : w \in \mathcal{N}_G(v)\}\!\} = \{\!\{(\chi_H^{(T)}(x,z), \deg_H(z)) : z \in \mathcal{N}_H(y)\}\!\}.$$

By definition of node marking policy, we further obtain

$$\{\!\{(\chi_G^{(T)}(u,w), \deg_G(w)) : w \in \mathcal{N}_G(v)\backslash\{u\}\}\!\} = \{\!\{(\chi_H^{(T)}(x,z), \deg_H(z)) : z \in \mathcal{N}_H(y)\backslash\{x\}\}\!\}.$$

By induction,

$$\begin{aligned}&\{\!\{(\deg_G(w), \{\!\{\omega(Q) : Q \in \mathcal{Q}_G^T(w,u)\}\!\}) : w \in \mathcal{N}_G(v)\backslash\{u\}\}\!\}\\ =&\{\!\{(\deg_H(z), \{\!\{\omega(Q) : Q \in \mathcal{Q}_H^T(z,x)\}\!\}) : z \in \mathcal{N}_H(y)\backslash\{x\}\}\!\}.\end{aligned}$$

Therefore, $\{\!\{\omega(Q) : Q \in \mathcal{Q}_G^{T+1}(v,u)\}\!\} = \{\!\{\omega(Q) : Q \in \mathcal{Q}_H^{T+1}(y,x)\}\!\}$, concluding the induction step. □

Corollary J.2. *Consider any SWL algorithm* $\mathsf{A}(\mathcal{A}, \mathsf{Pool})$ *with* $\mathsf{agg}_\mathsf{u}^\mathsf{L} \in \mathcal{A}$ *and let* χ *be the stable color mapping. For any vertices* $u, v \in \mathcal{V}_G$ *and* $x, y \in \mathcal{V}_H$*, if* $\chi_G(u,v) = \chi_H(x,y)$*, then* $\mathrm{dis}_G^\mathsf{H}(v,u) = \mathrm{dis}_H^\mathsf{H}(y,x)$.

Proof. By definition of hitting time distance,

$$\mathrm{dis}_G^\mathsf{H}(v,u) = \sum_{i=0}^{\infty} i \cdot \sum_{Q \in \mathcal{Q}_G^i(v,u)} 1/q(Q),$$

where $q(Q) = \deg_G(x_0)\prod_{i=1}^{d-1}\deg(x_i)$ for path $Q = (x_0, x_1, \cdots, x_d)$. Therefore, $q(Q)$ is fully determined by $\omega(Q)$ and $\deg_G(x_0)$. If $\mathrm{dis}_G^\mathsf{H}(v,u) \neq \mathrm{dis}_H^\mathsf{H}(y,x)$, then either $\deg_G(v) \neq \deg_H(y)$ or there exists a length t such that $\{\!\{\omega(Q) : Q \in \mathcal{Q}_G^t(v,u)\}\!\} \neq \{\!\{\omega(Q) : Q \in \mathcal{Q}_H^t(y,x)\}\!\}$. Therefore, by using Lemma J.1 we have $\chi_G(u,v) \neq \chi_H(x,y)$, as desired. □

We are now ready to prove the main theorem:

Theorem J.3. *Define a variant of* GD-WL *that incorporates the shortest path distance and the hitting time distance as follows:*

$$\chi_G^{(t+1)}(u) = \left\{\!\!\left\{ \left((\mathrm{dis}_G(v,u), \mathrm{dis}_G^\mathsf{H}(v,u)), \chi_G^{(t)}(v) \right) : v \in \mathcal{V}_G \right\}\!\!\right\}.$$

Then, GD-WL $\preceq$ PSWL(VS).

Proof. Denote χ^P as the stable color mapping of PSWL(VS). Consider a pair of graphs G and H indistinguishable by PSWL(VS). Then, we clearly have

$$\{\!\{\chi_G^\mathsf{P}(u,v) : u, v \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H^\mathsf{P}(x,y) : x, y \in \mathcal{V}_H\}\!\}.$$

By definition of the node marking policy,

$$\{\!\{\chi_G^\mathsf{P}(u,u) : u \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H^\mathsf{P}(x,x) : x \in \mathcal{V}_H\}\!\}.$$

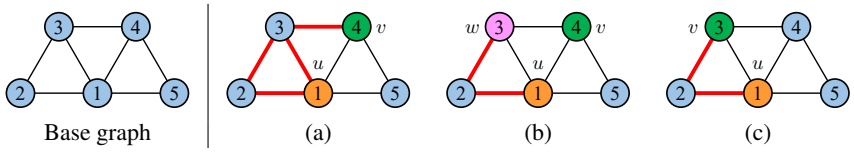


Figure 12. Illustration of the proof of Lemma J.6. When Duplicator follows her optimal strategy, the game process of SWL(VS) corresponds to figures (a, b, c) and Spoiler can eventually win.

Now consider any vertices $u \in \mathcal{V}$ and $x \in \mathcal{V}_H$ satisfying $\chi^{\mathsf{P}}_G(u,u) = \chi^{\mathsf{P}}_H(x,x)$. Since $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ is present in PSWL(VS), we can invoke Lemma E.6, which obtains that

$$\{\!\{\chi^{\mathsf{P}}_G(u,w) : w \in \mathcal{V}_G\}\!\} = \{\!\{\chi^{\mathsf{P}}_H(x,z) : z \in \mathcal{V}_H\}\!\}.$$

Further using Corollaries E.5 and J.2 yields

$$\{\!\{(\chi^{\mathsf{P}}_G(u,w), \mathrm{dis}_G(w,u), \mathrm{dis}^{\mathsf{H}}_G(w,u)) : w \in \mathcal{V}_G\}\!\} = \{\!\{(\chi^{\mathsf{P}}_H(x,z), \mathrm{dis}_G(z,x), \mathrm{dis}^{\mathsf{H}}_G(z,x)) : z \in \mathcal{V}_H\}\!\}.$$

Next, by definition of the aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$, we have

$$\{\!\{(\chi^{\mathsf{P}}_G(w,w), \mathrm{dis}_G(w,u), \mathrm{dis}^{\mathsf{H}}_G(w,u)) : w \in \mathcal{V}_G\}\!\} = \{\!\{(\chi^{\mathsf{P}}_H(z,z), \mathrm{dis}_G(z,x), \mathrm{dis}^{\mathsf{H}}_G(z,x)) : z \in \mathcal{V}_H\}\!\}.$$

The above equation shows that χ^{P}_G induces a finer *vertex* partition (i.e., by treating $\chi^{\mathsf{P}}_G(u) := \chi^{\mathsf{P}}_G(u,u)$) compared with the stable color mapping of GD-WL. Concretely, based on Remark E.2(c), we have $\chi^{\mathsf{P}}_G(u,u) = \chi^{\mathsf{P}}_H(x,x) \implies \chi_G(u) = \chi_H(x)$. This finally yields

$$\{\!\{\chi_G(u) : u \in \mathcal{V}_G\}\!\} = \{\!\{\chi_H(x) : x \in \mathcal{V}_H\}\!\},$$

concluding the proof. □

Remark J.4. We note that the form of GD-WL in Theorem J.3 slightly differs from Zhang et al. (2023), in that they use resistance distance instead of hitting time distance. Nevertheless, similar to resistance distance, hitting time distance also satisfies the following key property: for any vertices $u, v, w \in \mathcal{V}_G$ in graph G, $\mathrm{dis}^{\mathsf{H}}_G(u,v) = \mathrm{dis}^{\mathsf{H}}_G(u,w) + \mathrm{dis}^{\mathsf{H}}_G(w,v)$ if and only if w is a cut vertex of G (see Zhang et al. (2023, Appendix C.5.1)). This property is crucial to prove the expressivity for vertex-biconnectivity. Following the almost same proof, we can show that the variant of GD-WL defined in Theorem J.3 is also fully expressive for vertex-biconnectivity.

Finally, for the original GD-WL defined in Zhang et al. (2023) that incorporates SPD and RD, currently we can only prove the following result, which is a straightforward extension of Theorem J.3:

Theorem J.5. *Consider the WL algorithm* GD-WL *that incorporates the shortest path distance and the resistance distance. Then,* GD-WL $\preceq$ SSWL.

Proof. Note that $\mathrm{dis}^{\mathsf{R}}_G(u,v) = (\mathrm{dis}^{\mathsf{H}}_G(u,v) + \mathrm{dis}^{\mathsf{H}}_G(v,u))/2|\mathcal{E}_G|$. The proof follows by noting that both $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$ and $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$ are present in SSWL. □

However, it remains unclear whether RD-WL $\preceq$ GSWL or RD-WL $\preceq$ PSWL(VS) holds. We leave them as open problems.

J.2. Counterexamples

We first show that the vanilla SWL has additional power than GD-WL.

Lemma J.6. *There exist two non-isomorphic graphs such that*

- SWL(VS) *can distinguish them;*
- RD-WL, HTD-WL, SPD-WL *cannot distinguish them.*

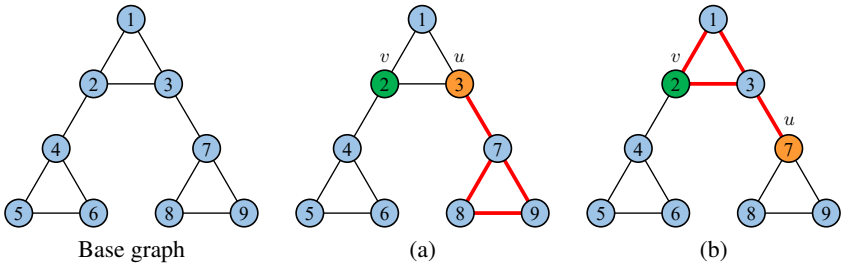


Figure 13. Illustration of the proof of Lemma J.7. For SWL(SV), when Duplicator follows her optimal strategy, Spoiler can never win the game.

Proof. The proof is based on Appendix I.3 using generalized Fürer graphs. The base graph is constructed in Figure 12.

We first consider SWL(VS) and analyze the simplified pebbling game developed in Appendix I.2. At the beginning, Spoiler just places pebble u on vertex 1, and Duplicator does nothing. Spoiler then places pebble v on vertex 4, splitting the connected component into three parts. It is easy to see that Duplicator should select the largest connected component on the left (Figure 12(a)). In the next round, Spoiler places pebbles w adjacent to pebble v on vertex 3. Duplicator has to respond according to Figure 12(b). Spoiler then swaps pebbles v, w and leaves w outside the graph (Figure 12(c)). It is easy to see that Spoiler can win the remaining game.

We next consider GD-WL, for which we do not have a corresponding game. Nevertheless, a good news is that the corresponding (twisted) Fürer graph has only 20 vertices. We can thus directly verify that the stable colors of the Fürer graph match those of the twisted Fürer graph. A deep understanding of why GD-WL cannot distinguish the two graphs is left for future work. □

Conversely, we next show that GD-WL also has additional power than the vanilla SWL.

Lemma J.7. *There exist two non-isomorphic graphs such that*

- SWL(SV) *cannot distinguish them;*
- SPD-WL *can distinguish them;*
- RD-WL *can distinguish them;*
- HTD-WL *can distinguish them.*

Proof. The proof is based on Appendix I.3 using generalized Fürer graphs. The base graph is constructed in Figure 13.

We first consider SWL(SV) and analyze the simplified pebbling game developed in Appendix I.2. At the beginning, Spoiler should place pebble v on some vertex. Regardless of Spoiler's choice, Duplicator just selects the largest connected component after pebble v is placed. Next, Spoiler should place pebble u on some vertex. Now Duplicator's strategy is to select a connected component such that it contains a triangle with no pebbles. It is easy to see that Duplicator can always achieve her goal (see Figure 13(a) and Figure 13(b) for two representative cases). The remaining game is easy to analyze: since pebble u cannot be moved throughout the game, there is a triangle that holds at most two pebbles and cannot be split into three single-edge components. Clearly, Duplicator can always respond without losing the game.

We next turn to SPD-WL, for which we do not have a corresponding game. Nevertheless, a good news is that the corresponding (twisted) Fürer graph has only 26 vertices. We can thus directly verify that the stable colors of the Fürer graph do not match those of the twisted Fürer graph. The case of RD-WL and HTD-WL can be similarly verified. A deep understanding of why these algorithms can distinguish the two graphs is left for future work. □

K. Experimental Details

We conduct experiments on three standard benchmark datasets: ZINC (Dwivedi et al., 2020), Counting Substructure (Zhao et al., 2022a; Frasca et al., 2022), and OGBG-molhiv (Hu et al., 2020). ZINC is a standard benchmark for molecular property prediction, where the task is to predict the constrained solubility of a molecule, which is an important chemical

property for drug discovery. We train and evaluate our proposed GNN-SSWL and GNN-SSWL+ on both ZINC (consisting of 250k molecular graph) and ZINC-subset (a 12K-subset selected as in Dwivedi et al. (2020)). Counting Substructure is a widely-used synthetic task in the expressive GNN community, where the task is to predict the number of a given substructure (such as cycle or star) in an input graph. We use the same dataset in Zhao et al. (2022a); Frasca et al. (2022) and further extend it to include the setting of counting 5/6-cycles motivated by Huang et al. (2022). Finally, we additionally consider the OGBG-molhiv dataset in Appendix K.4.

K.1. Model details

We implement our model using Pytorch (Paszke et al., 2019) and Pytorch Geometric (Fey & Lenssen, 2019) (available respectively under the BSD and MIT license). All experiments are run on a single NVIDIA Tesla V100 GPU. Our code will be released at `https://github.com/subgraph23/SWL`.

Motivated by Propositions 4.2 and E.3, for all SWL models, the graph generation policy is chosen as the distance encoding on the original graph. Such a policy achieves the maximal power among other policies (as expressive as node marking) while explicitly introducing inductive biases, which may be beneficial for real-world tasks. Concretely, we initialize the feature of node v in subgraph G^u by summing the atom embedding $h^{\mathsf{atom}}(v)$ and the distance encoding $h^{\mathsf{dis}}(\mathrm{dis}_G(u,v))$, where the atom embedding is a learnable vector determined by the atom type of v, and the distance encoding is a learnable vector determined by the shortest path distance between u and v. Mathematically, $h_G^{(0)}(u,v) = h^{\mathsf{atom}}(v) + h^{\mathsf{dis}}(\mathrm{dis}_G(u,v))$. Distances exceeding `max_dis` (including infinity) are all encoded as a shared embedding. For tasks without atom features (e.g., Counting Substructure dataset), we set $h^{\mathsf{atom}}(v)$ to zero.

As an instance of Definition 2.1, our subgraph GNN layer can be written in the following form:

$$h_G^{(l+1)}(u,v) = \mathsf{ReLU}\left(\sum_{i=1}^{r} \mu^{(l+1,i)}\left(h_G^{(l)}(u,v), \mathsf{op}_i(u,v,G,h_G^{(l)})\right)\right), \tag{25}$$

where each op_i can take one of the following forms, depending on the atomic aggregations in the SWL algorithm:

- For $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uu}}$: $\mathsf{op}_i(u,v,G,h) = h(u,u)$;
- For $\mathsf{agg}^{\mathsf{P}}_{\mathsf{vv}}$: $\mathsf{op}_i(u,v,G,h) = h(v,v)$;
- For $\mathsf{agg}^{\mathsf{G}}_{\mathsf{u}}$: $\mathsf{op}_i(u,v,G,h) = \sum_{w\in\mathcal{V}_G} h(u,w)$;
- For $\mathsf{agg}^{\mathsf{G}}_{\mathsf{v}}$: $\mathsf{op}_i(u,v,G,h) = \sum_{w\in\mathcal{V}_G} h(w,v)$;
- For $\mathsf{agg}^{\mathsf{L}}_{\mathsf{u}}$: $\mathsf{op}_i(u,v,G,h) = \sum_{w\in\mathcal{N}_G(v)} \mathsf{ReLU}((h(u,w)+g(w,v))$;
- For $\mathsf{agg}^{\mathsf{L}}_{\mathsf{v}}$: $\mathsf{op}_i(u,v,G,h) = \sum_{w\in\mathcal{N}_G(u)} \mathsf{ReLU}(h(w,v)+g(u,w))$.

Note that we have included the single-point aggregation $\mathsf{agg}^{\mathsf{P}}_{\mathsf{uv}}$ directly in the update formula (25). For the last two local aggregations, we further encode the edge embedding $g(w,v)$ for edge $\{w,v\}\in\mathcal{E}_G$ (or $g(u,w)$ for edge $\{w,u\}\in\mathcal{E}_G$) when there is additional information for each edge (e.g., the bond information in the ZINC dataset). In the above equation, each $\mu^{(l+1,i)}$ is implemented by a GIN base encoder (Xu et al., 2019):

$$\mu^{(l+1,i)}(h_1,h_2) = \mathsf{MLP}^{(l+1,i)}\left((1+\epsilon^{(l+1,i)})h_1 + h_2\right),$$

where $\epsilon^{(l+1,i)}$ is a learnable scalar and $\mathsf{MLP}^{(l+1,i)}$ is a one-hidden-layer MLP with hidden size equal to the input dimension. Batch Normalization (Ioffe & Szegedy, 2015) is adopted in the hidden layer of each MLP as well as in (25) before taking ReLU.

The final pooling layer is implemented as an MLP over the summation of v-dimension for VS-pooling scheme (or u for SV-pooling) and a global mean pooling, namely

$$f(G) = \frac{1}{|\mathcal{V}_G|}\sum_{u\in\mathcal{V}_G} \mathsf{MLP}\left(\sum_{v\in\mathcal{V}_G} h_G^{(L)}(u,v)\right)$$

Batch Normalization is adopted similarly.

K.2. Training details

ZINC. Throughout all experiments, we set the number of layers $L = 6$, similar to Frasca et al. (2022). To constrain the parameter budget within 500k, the feature dimension of each layer is set to 96. The initial atom embedding, distance embedding, and edge embedding is also set to 96. The hyper-parameter `max_dis` is set to 5. We adopt the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate 0.001. The learning rate will be decayed by a factor of 0.5 when the MAE on *validation set* plateaus for 20 epochs (similar to Frasca et al. (2022)). The batch size is set to 128. On ZINC-12K subset, the model is trained for 400 epochs according to Frasca et al. (2022), and it takes roughly 1 to 2 hours for a single run. On ZINC-250K full set, we find that the model still does not converge after 400 epochs, so we adjust the configuration to 500 epochs, which takes about 40 hours. For each setting, we run the model 10 times with different seeds from 1 to 10 and report both the mean value and the standard deviation of MAE.

We also compare our model performance with various subgraph GNN baselines. The performance numbers of these baselines in Table 2 are generally brought from the original works. For some baselines such that GNN-AK and GNN-AK-ctx, the results are obtained from Frasca et al. (2022). The NGNN result is obtained from Huang et al. (2022). Since other baseline models did not present the results on ZINC-full, we obtain their performance by running the code provided on the authors' official GitHub repo. We run each model 10 times with different seeds and report the mean performance and standard deviation. For ESAN and SUN, we tried both the k-ego network policy and the k-ego network policy with marking and reported the better performance among the two policies. We find that the results are almost the same for SUN, and the k-ego network policy with marking is slightly better for ESAN.

Counting Substructure. Throughout all experiments, we simply follow the same model configuration as ZINC to use a 6-layer GNN with a hidden size of 96. The hyper-parameter `max_dis` is also set to 5. We adopt the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate 0.002. The learning rate is decayed with cosine annealing. The batch size is set to 512. The models are trained for 600 epochs. Note that unlike prior works, *we use the same model/training hyper-parameters for all six substructures*. For each setting, we run the model 5 times with different seeds from 1 to 5 and report the mean performance of MAE. We found that the standard deviation is very small.

We also compare our model performance with various subgraph GNN baselines. The performance numbers of GNN-AK and SUN in Table 1 are brought from the Zhao et al. (2022a) and Frasca et al. (2022), respectively. For the tasks of counting 5/6-cycles, we obtain their performance by running the code provided on the authors' official GitHub repo. To make the SUN result convincing, we grid search the network width from $\{64, 96, 110\}$, depth from $\{5, 6\}$, and search the k-hop ego net with $k \in \{2, 3\}$ as suggested by Frasca et al. (2022). We consider both the ego network policy with and without marking and find that k-ego network with marking is better.

K.3. Ablation study on ZINC

In this subsection, we present a set of ablation results to investigate the effect of different aggregation operations in GNN-SSWL+. We fix all model details and hyper-parameters presented above, and remove one or both of the additional two operations $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}$ in GNN-SSWL+ or change the pooling paradigm. This results in several types of models with expressivity corresponding to SSWL, PSWL(VS), PSWL(SV), SWL(SV), and SWL(VS), respectively. The results are shown in Table 3. It can be seen that both of these additional aggregations $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}, \mathsf{agg}_{\mathsf{vv}}^{\mathsf{P}}$ provide a significant improvement in performance. Moreover, SV pooling is significantly better than VS for vanilla SWL.

We further design an ablation experiment to verify that our introduced local aggregation $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$ is actually crucial and cannot be replaced by other aggregations. Here, we consider the model SUN proposed in Frasca et al. (2022), which comprises a large number of basic aggregation operations but without $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$. SUN uses different parameters to compute the on-diagonal (i.e. $h_G^{(l)}(u, u)$) and off-diagonal (i.e. $h_G^{(l)}(u, v), v \neq u$) features in order to further enhance the model flexibility. We rerun their code by changing the graph generation policy to distance encoding (on the original graph) and use exactly the same training configuration as our model. Notably, the feature dimension is increased to 96 (instead of 64), resulting in a larger model with roughly 1163k parameters. We can see in Table 3 that, even with distance encoding policy and larger model size, there is still a remarkable gap between SUN and GNN-SSWL+ (less than 400k parameters). This confirms that the introduced aggregation $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$ in SSWL not only theoretically improves the expressive power of the GNN model, but also leads to significantly better performance on real datasets.

Table 3. Ablation study of GNN-SSWL+ on ZINC-subset.

Method	Pooling	Test MAE ↓
GNN-SSWL+	VS	**0.0703 ± 0.0046**
w/o $\mathsf{agg}_{\mathsf{vv}}$ (GNN-SSWL)	VS	0.0822 ± 0.0029
w/o $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$	VS	0.0765 ± 0.0028
w/o $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$	SV	0.0758 ± 0.0037
w/o $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$ and $\mathsf{agg}_{\mathsf{vv}}$	VS	0.1103 ± 0.0090
w/o $\mathsf{agg}_{\mathsf{v}}^{\mathsf{L}}$ and $\mathsf{agg}_{\mathsf{vv}}$	SV	0.0999 ± 0.0044
SUN (Distance Encoding)	-	0.0802 ± 0.0024

K.4. Additional experiments on OGBG-molhiv

We further run experiments on the OGBG-molhiv dataset. Following Frasca et al. (2022), we use a 2-layer GNN-SSWL+ model with a network width of 64, and add residual connection between different layers. The hyper-parameter `max_dis` is also set to 5. To prevent overfitting, we similarly use the ASAM optimizer (Kwon et al., 2021) with a batch size of 32, a learning rate of 0.01, and a dropout ratio of 0.3. Moreover, we change each MLP to a linear layer following Frasca et al. (2022). We train the model for 100 epochs. We run our model 8 times with different seeds ranging from 1 to 8 and report the average ROC AUC as well as the standard deviation. The result is presented in Table 4.

Table 4. Performance comparison on OGBG-molhiv.

Model	Reference	Test ROC-AUC (%)
GCN	Kipf & Welling (2017)	76.06±0.97
GIN	Xu et al. (2019)	75.58±1.40
PNA	Corso et al. (2020)	79.05±1.32
GSN	Bouritsas et al. (2022)	80.39±0.90
CIN	Bodnar et al. (2021a)	**80.94±0.57**
Recon. GNN	Cotta et al. (2021)	76.32±1.40
DS-GNN (EGO+)	Bevilacqua et al. (2022)	77.40±2.19
DSS-GNN (EGO+)	Bevilacqua et al. (2022)	76.78±1.66
GNN-AK+	Zhao et al. (2022a)	79.61±1.19
SUN	Frasca et al. (2022)	80.03±0.55
GNN-SSWL+	This paper	79.58±0.35