Local Vertex Colouring Graph Neural Networks

Shouheng Li [1 2] **Dongwoo Kim** [3] **Qing Wang** [1]

Abstract

In recent years, there has been a significant amount of research focused on expanding the expressivity of Graph Neural Networks (GNNs) beyond the Weisfeiler-Lehman (1-WL) framework. While many of these studies have yielded advancements in expressivity, they have frequently come at the expense of decreased efficiency or have been restricted to specific types of graphs. In this study, we investigate the expressivity of GNNs from the perspective of graph search. Specifically, we propose a new vertex colouring scheme and demonstrate that classical search algorithms can efficiently compute graph representations that extend beyond the 1-WL. We show the colouring scheme inherits useful properties from graph search that can help solve problems like graph biconnectivity. Furthermore, we show that under certain conditions, the expressivity of GNNs increases hierarchically with the radius of the search neighbourhood. To further investigate the proposed scheme, we develop a new type of GNN based on two search strategies, breadth-first search and depth-first search, highlighting the graph properties they can capture on top of 1-WL. Our code is available at https://github.com/seanli3/lvc.

1. Introduction

Graph neural networks (GNNs) have emerged as the de-facto method for representation learning on graphs. One popular architecture of GNNs is the message-passing neural networks (MPNNs) which propagate information between vertices along edges (Gilmer et al., 2017; Kipf & Welling, 2017; Velickovic et al., 2017). In Xu et al. (2019), it is shown that the design of MPNNs aligns with the Weisfeiler-Lehman (1-WL) test, a classical algorithm for testing graph isomorphism. Therefore, the expressivity of MPNNs is upper-bounded by the 1-WL test. Intuitively, if two vertices have the same computational/message-passing graph in an MPNN, they are indistinguishable.

Recent studies attempted to increase the expressivity of GNNs beyond the 1-WL. One direction is to extend GNNs to match higher-order WL tests (Morris et al., 2019; 2020; Maron et al., 2019; Geerts & Reutter, 2022). While these methods offer improved expressivity, they come at the cost of decreased efficiency, as higher-order WL tests are known to be computationally expensive. Another line of research focuses on incorporating graph substructures into feature aggregation (Bodnar et al., 2021b;a). However, these approaches often rely on task-specific, hand-picked substructures. One other strategy is to enhance vertex or edge features with additional distance information relative to target vertices (You et al., 2019; Li et al., 2020). Despite the efforts, Zhang et al. (2023) have shown that MPNNs cannot solve the biconnectivity problem, which can be efficiently solved using the depth-first search algorithm (Tarjan, 1974).

In light of the aforementioned understanding, one may question how the design of GNNs can surpass the limitations of 1-WL to address issues that cannot be resolved by MPNNs. In this work, we systematically study an alternative approach to MPNN, which propagates information along graph search trees. This paper makes the following contributions:

- We design a novel colouring scheme, called *local vertex colouring* (LVC), based on breath-first and depth-first search algorithms, that goes beyond 1-WL.
- We show that LVC can learn representations to distinguish several graph properties such as biconnectivity, cycles, cut vertices and edges, and *ego short-path graphs* (ESPGs) that 1-WL and MPNNs cannot.
- We analyse the expressivity of LVC in terms of *breadth-first colouring* and *depth-first colouring*, and provide systematical comparisons with 1-WL and 3-WL.
- We further design a graph search-guided GNN architecture, *Search-guided Graph Neural Network* (SGN), which inherits the properties of LVC.

[1]School of Computing, Australian National University, Canberra, Australia [2]Data61, CSIRO, Canberra, Australia [3]CSE & GSAI, POSTECH, Pohang, South Korea. Correspondence to: Dongwoo Kim <dongwoo.kim@postech.ac.kr>, Qing Wang <qing.wang@anu.edu.au>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

2. Related Work

Graph isomorphism and colour refinement. The graph isomorphism problem concerns whether two graphs are identical topologically, e.g. for two graphs G and H, whether there is a bijection $f : V_G \to V_H$ such that any two vertices u and v are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. If such a bijection exists, we say G and H are *isomorphic* ($G \simeq H$).

Let C be a set of colours. A *vertex colouring refinement* function $\lambda : V \to C$ assigns each vertex $v \in V$ with a colour $\lambda(v) \in C$. This assignment is performed iteratively until the vertex colours no longer change. Colour refinement can be used to test graph isomorphism by comparing the multisets of vertex colours $\{\!\{\lambda(v) : v \in V_G\}\!\}$ and $\{\!\{\lambda(u) : u \in V_H\}\!\}$, given that $\lambda(\cdot)$ is invariant under isomorphic permutations. A classic example of such tests is the *Weisfeiler-Lehman (WL) test* (Weisfeiler & Leman, 1968), which assigns a colour to a vertex based on the colours of its neighbours. Cai et al. (1992) extends 1-WL to compute a colour on each k-tuple of vertices; this extension is known as the *k-dimensional Folklore Weisfeiler-Lehman algorithms* (k-FWL). We may apply a colouring refinement to all vertices in G iteratively until vertex colours are stabilised.

Let (G, λ^i) denote a colouring on vertices of G after applying a colour refinement function i times, i.e., after the i-th iteration, and $P(\lambda^i)$ a partition of the vertex set induced by the colouring (G, λ^i). For two vertex partitions $P(\lambda^i)$ and $P(\lambda^j)$ on G, if every element of $P(\lambda^i)$ is a (not necessarily proper) subset of an element of $P(\lambda^j)$, we say $P(\lambda^i)$ *refines* $P(\lambda^j)$. When $P(\lambda^j) \equiv P(\lambda^{j+1})$, we call λ^j a *stable colouring* and $P(\lambda^j)$ a *stable partition* of G.

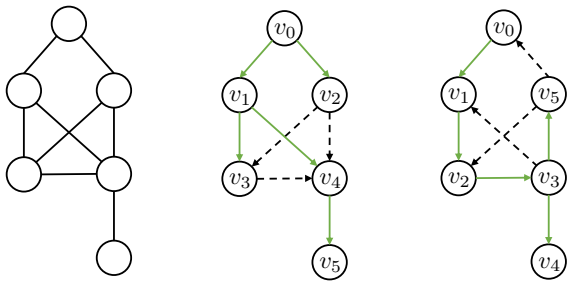
GNNs beyond 1-WL. MPNN is a widely adopted graph representation learning approach in many applications. However, there are a few caveats. Firstly, as shown by Xu et al. (2019), the expressive power of MPNN is upper-bounded by 1-WL, which is known to have limited power in distinguishing isomorphic graphs. Secondly, to increase the receptive field, MPNN needs to be stacked deeply, which causes over-smoothing (Zhao & Akoglu, 2019; Chen et al., 2020) and over-squashing (Topping et al., 2022) that degrade performance. Recent research aims to design more powerful GNNs by incorporating higher-order neighbourhoods (Maron et al., 2019; Morris et al., 2019). However, these methods incur high computational costs and thus are not feasible for large datasets. Other methods alter the MPNN framework or introduce extra heuristics to improve expressivity (Bouritsas et al., 2022; Bodnar et al., 2021b;a; Bevilacqua et al., 2022; Wijesinghe & Wang, 2022). However, while these methods are shown to be more powerful than 1-WL, it is still unclear what additional properties they can capture beyond 1-WL.

GNNs in learning graph algorithms. Velickovic et al. (2020) show that MPNN can imitate classical graph algorithms to learn shortest paths (Bellman-Ford algorithm) and minimum spanning trees (Prim's algorithm). Georgiev & Lió (2020) show that MPNNs can execute the more complex Ford-Fulkerson algorithm, which consists of several composable subroutines, for finding maximum flow. Loukas (2020) further shows that certain GNN can solve graph problems like cycle detection and minimum cut, but only when its depth and width reach a certain level. Xu et al. (2020) show that the ability of MPNN to imitate complex graph algorithms is restrained by the alignment between its computation structure and the algorithmic structure of the relevant reasoning process. One such example, as shown by Zhang et al. (2023), is that MPNN cannot solve the biconnectivity problem, despite that this problem has an efficient algorithmic solution linear to graph size.

3. Preliminaries

Let $\{\cdot\}$ denote sets and $\{\!\{\cdot\}\!\}$ multisets. We consider undirected simple graphs $G = (V, E)$ where V is the vertex set and E is the edge set. We use $|\cdot|$ to denote the cardinality of a set/multiset/sequence, $e_{vu} = \{v, u\}$ an undirected edge connecting vertices v and u, and $\vec{e}_{vu} = (v, u)$ a directed edge that starts from vertex v and ends at vertex u. Thus, $e_{vu} = e_{uv}$ and $\vec{e}_{vu} \neq \vec{e}_{uv}$. We use $d(v, u)$ to denote the shortest-path distance between vertices v and u. A *δ-neighborhood* of a vertex v is a set of vertices within the distance δ from v, i.e. $N_\delta(v) = \{u \in V : 1 \leq d(v, u) \leq \delta\}$. A path $\mathcal{P}_{w_0 w_k}$ of length k in G, called *k-path*, is a sequence $(w_0, w_1, ..., w_k)$ of distinct vertices such that $(w_{i-1}, w_i) \in E$ for $i = 1, 2, ..., k$.

Graph searching. Graph traversal, or *graph searching*, visits each vertex in a graph. *Breadth-First Search* (BFS) and *Depth-First Search* (DFS) are the two most widely used graph search algorithms, which differ in the order of visiting vertices. Both methods start at a vertex v. BFS first visits the direct neighbours in $N_1(v)$ of v and then the neighbours in $N_2(v)$ that have not been visited, etc. The idea is to process all vertices in N_i of distance (or level) i from v before processing vertices at level $i + 1$ or greater. The process is repeated until all vertices reachable from v have been visited. In DFS we instead go as "deep" as possible from vertex v until no new vertices can be visited. Then we backtrack and try other neighbours that were missed from the farthest in the search paths until all vertices are visited. The visited vertices and the edges along the search paths form a BFS/DFS search tree. For example, the solid lines in Figures 1b and 1c represent two different search trees for the graph in Figure 1a. Given a graph, generally, there are many ways to construct different search trees by selecting different starting points and different edges to visit vertices.



(a) (b) BFS (c) DFS

Figure 1: Tree edges (green solid lines) and back edges (black dashed lines) classified by BFS and DFS. The subscripted labels $v_0, \ldots, v_5$ denote the visit sequence of each vertex, e.g. v_1 is visited after v_0.

Visiting order subscripting. Vertices being visited in a graph search form a linear order. We use subscripts to label this order based on their discovery/first-visited time (Cormen et al., 2022). Given a search tree, we say v_i *precedes* v_j, denoted as $v_i \prec v_j$, if v_i is discovered before v_j. i and j are subscripts that indicate the relation: for all i and j, $v_i \prec v_j$ if and only if $i < j$. Each vertex is discovered once, so a subscript ranges from 0 to $N-1$ in a graph G. v_0 is called the root. Figure 1 shows how the subscripting is used to indicate vertex visiting orders.

Tree and back edges. For an undirected simple graph $G = (V, E)$, BFS/DFS categorise the edges in E into *tree edges* and *back edges*.

Tree edges and back edges are both directed, i.e., $(v, u) \neq (u, v)$. Let T_v denote a search tree rooted at vertex v. A tree edge is an edge in the search tree T_v, starting from an early-visited vertex to a later-visited vertex. A back edge starts from a later-visited vertex to an early-visited vertex. A directed edge (v_i, v_j) is either a tree edge (if $i < j$), or a back edge (if $i > j$). For example, dashed lines in Figures 1b and 1c represent back edges in BFS and DFS search trees, respectively. In BFS, a back edge (v_i, v_j) connects vertices at the same or adjacent level, i.e. $d(v_0, v_i) = d(v_0, v_j)$ or $|d(v_0, v_i) - d(v_0, v_j)| = 1$ (Cormen et al., 2022). For this reason back edges in BFS are sometimes referred to as *cross edges*. In DFS, a back edge (v_i, v_j) connects a vertex v_i and its non-parent ancestor v_j; therefore we always have $v_j \prec v_i$ (or $i > j$).

We use $E^{T_v}_{\text{tree}}$ and $E^{T_v}_{\text{back}}$ to denote the tree edge set and the back edge set with respect to a search tree T_v rooted at v.

4. Local Vertex Colouring

In this section, we introduce the building blocks of a search-guided colouring scheme and demonstrate its expressive power in distinguishing non-isomorphic graphs. We also discuss how this search-guided colouring scheme can solve the biconnectivity and ego shortest-path graph problems that cannot be solved by MPNN.

4.1. Search-guided Vertex Colour Update

We first describe a way to update vertex colours guided by graph searching. As described in Section 3, graph searching w.r.t. a search tree T_v categorises edges into two sets: a tree edge set $E^{T_v}_{\text{tree}}$ and a back edge set $E^{T_v}_{\text{back}}$. Instead of propagating messages along all edges like MPNN, we design a scheme that propagates vertex information, i.e., vertex colours, along tree edges and back edges. Given a search tree T_v, we define the vertex colouring refinement function on each vertex u as follows

$$\lambda^{l+1}(u) := \rho(\lambda^l(u), \{\!\{\lambda^{l+1}_v(u) : v \in N_\delta(u)\}\!\}) \quad (1)$$

where $\rho(\cdot)$ is an injective function and $\lambda^{l+1}_v(u)$ is the vertex colour computed, based on the search tree T_v, as

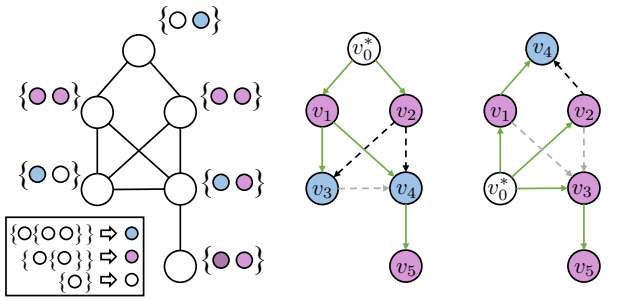
$$\lambda^{l+1}_v(u) := \phi\left(\lambda^l(u), \psi\left(\{\!\{\lambda^l_v(w) : w \in \eta(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})\}\!\}\right)\right) \quad (2)$$

where $\phi(\cdot)$ and $\psi(\cdot)$ are injective functions, and l is the number of the current iteration. Let $\mathbb{P}(E)$ and $\mathbb{P}(V)$ be the power sets of E and V, respectively. The function $\eta : V \times \mathbb{P}(E) \times \mathbb{P}(E) \to \mathbb{P}(V)$ takes a vertex $u \in V$ to be coloured, a tree edge set $E^{T_v}_{\text{tree}} \in \mathbb{P}(E)$, and a back edge set $E^{T_v}_{\text{back}} \in \mathbb{P}(E)$ as input, and produces a vertex set in $\mathbb{P}(V)$.

At the first iteration, $\lambda^0_v(w)$ and $\lambda^0(w)$ are the initial colours of w. There are two steps in the colouring scheme. The first step is *search-guided colour propagation* (Equation (2)), where vertex colours are propagated along the search paths of each T_v based on $\eta(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})$. After this step, a vertex u obtains a colour $\lambda^{l+1}_v(u)$ w.r.t. each root vertex $v \in N_\delta(u)$. We obtain in total $|N_\delta(u)|$ colours for u. The second step is *neighbourhood aggregation* (Equation (1)), where the $|N_\delta(u)|$ colours obtained from the first step are aggregated and used to compute a new colour $\lambda^{l+1}(u)$ for u. These two steps are repeated with the new vertex colours.

When $N_\delta(v) \neq V$, we call this vertex colouring scheme δ-*local vertex colouring* (LVC-δ). LVC-δ is applied to colour vertices in G iteratively until vertex colours are stabilised. We omit superscript and use λ to refer to a stable colouring.

Search order permutation. Graph searching may encounter cases where a search algorithm needs to decide a priority between two or more unvisited vertices (a tie). In such cases, the search algorithm can visit any vertex in the tie. This yields different vertex visiting orders, we call this permutation in vertex visiting order *search order permutation*. For example, in Figure 1b, the BFS rooted at vertex v_0



(a) (b) (c)

Figure 2: A graph and its vertex colours after the first BFC iteration. 2a shows the uncoloured graph, where the vertex colours obtained from 2b and 2c are shown next to each vertex. 2b and 2c show BFC with two different roots (marked with *), respectively, where each vertex is assigned a new colour by BFC. The colour map is shown in the bottom left corner. The subscripted labels $v_0, \ldots, v_5$ denote the visit sequence of each vertex, e.g. v_1 is visited after v_0. For brevity, we show BFC with only two roots.

faces a tie, where it can visit either v_1 or v_2 first since both v_1 and v_2 are adjacent to v_0. Figure 1b shows the search trajectory when visiting v_1 first; however, if the BFS visits v_2 first, then tree edges and back edges will be different.

Design considerations. A key function that controls the colouring in Equation 2 is η. For instance, we can make LVC-δ identical to 1-WL, if we define $\eta(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}}) = \{w : (u,w) \in E^{T_v}_{\text{tree}} \vee (w,u) \in E^{T_v}_{\text{tree}} \vee (u,w) \in E^{T_v}_{\text{back}} \vee (w,u) \in E^{T_v}_{\text{back}}\}$. We name it *1-WL-equivalent LVC*. In this definition, η effectively yields all neighbouring vertices of u, making it equivalent to 1-WL.

Since the design of η is crucial, we hereby introduce two key points that should be considered when designing η.

- η should be invariant to search order permutation, i.e. a change of vertex visiting order should not alter the output of η. If η is not invariant to search order permutation, the colouring scheme will not be permutation invariant. The 1-WL equivalent LVC example in the previous paragraph is invariant to search order permutation.
- η should inherit properties of a search algorithm that are informative about identifying graph structure. Graph searches like BFS and DFS are widely used in graph algorithms to capture structural properties such as cycles and biconnectivity. η should be designed to incorporate such structural information in vertex colours.

4.2. BFS-guided Colouring

BFS is perhaps the most widely used graph search algorithm and its applications include finding shortest paths, minimum spanning tree, cycle detection, and bipartite graph test. We design η for BFS, denoted as η_{bfc}, as

$$\eta_{\text{bfc}}(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}}) = \left\{o : (o,u) \in E^{T_v}_{\text{tree}} \vee \Big((o,u) \in E^{T_v}_{\text{back}} \wedge d(v,o) \neq d(v,u)\Big)\right\} \tag{3}$$

The part $\{o : (o,u) \in E^{T_v}_{\text{tree}}\}$ preserves vertices that lead to vertex u via tree edges. $\{o : (o,u) \in E^{T_v}_{\text{back}}\}$ preserves vertices that lead to vertex u via back edges. The condition $d(v,o) \neq d(v,u)$ ensures only back edges connecting vertices at different levels are included. The vertex colouring scheme using η_{bfc} is called *breadth-first colouring* (BFC).

Figure 2 shows BFC rooted at two different vertices, where the grey dashed lines indicate back edges that are excluded by BFC. In Figure 2b, when $u = v_4$, η_{bfc} returns v_1 and v_2, but excludes v_3 because it is at the same level as v_4.

An important property of BFS is that tree edges form the shortest paths between a root to other vertices, e.g., in Figure 2b (v_0, v_1) and (v_1, v_4) form the shortest path (v_0, v_1, v_4) between v_0 and v_4. There are two categories of back edges in a BFS tree: the ones connecting vertices across two adjacent levels and the ones connecting vertices at the same level (Cormen et al., 2022). The first category of back edges forms alternative shortest paths, e.g., (v_0, v_2) and (v_2, v_4) form another shortest path (v_0, v_2, v_4) between v_0 and v_4. The second category of back edges does not participate in any shortest path. η_{bfc} only preserves the first category of back edges and excludes the second category by the condition $d(v,o) \neq d(v,u)$. All shortest paths from a root to a vertex, e.g., (v_0, v_1, v_4) and (v_0, v_2, v_4), form an induced shortest path graph (SPG) (Wang et al., 2021), which is permutation invariant. For example, if we swap the search order between v_1 and v_2 in Figure 2b, (v_1, v_4) becomes a back edge and (v_2, v_4) becomes a tree edge, but η_{bfc} returns the same vertex set. Hence, BFC defined by Equations 1, 2, and 3 is also permutation invariant.

BFC is referred as BFC-δ if the search range is limited to a δ-hop neighbourhood for each root vertex.

Distinguishing shortest-path graphs. Velickovic et al. (2020) show that MPNN can imitate classical graph algorithms to learn shortest paths (Bellman-Ford algorithm) and minimum spanning trees (Prim's algorithm). Xu et al. (2020) further show that MPNN is theoretically suitable for learning tasks that are solvable using dynamic programming. Since the base form of BFC (i.e. BFC-1) aligns with MPNN (will be discussed later), BFC also inherits these properties. However, we are more interested in tasks which BFC can

do but MPNN cannot. We show that one of such tasks is to distinguish ego shortest-path graphs.

Definition 4.1. *For two vertices $v, u \in V_G$, a* shortest-path graph *$SPG(v, u)$ is a subgraph of G, where $SPG(v, u)$ contains all and only vertices and edges occurring in the shortest paths between u and v.*

Lemma 4.1. *Let (u, v) and (u', v') be two pairs of vertices. Then $SPG(u, v) \simeq SPG(u', v')$ if and only if one of the following conditions hold under BFC: (1) $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$; (2) $\lambda_v(u) = \lambda_{u'}(v')$ and $\lambda_u(v) = \lambda_{v'}(u')$.*

Definition 4.2. *Given a vertex $v \in V_G$ and a fixed $\delta \geq 1$, an* ego shortest-path graph *(ESPG) $S_v = (V_{S_v}, E_{S_v})$ is a subgraph of G, where $V_{S_v} = N_\delta(v)$ and E_S is the set of all edges that form all shortest paths between v and $u \in N_\delta(v)$.*

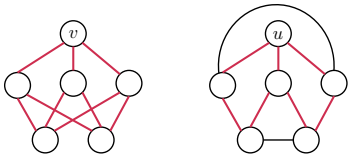


Figure 3: A pair of non-isomorphic three-regular graphs. Pink edges form ESPGs ($\delta = 2$) for vertices v and u.

Lemma 4.2. *Let $v, u \in V$ be any two vertices and $\lambda(v)$ and $\lambda(u)$ be the corresponding stable colours of v and u by running BFC. We have $S_v \simeq S_u$ if and only if $\lambda(v) = \lambda(u)$, where S_v and S_u are the ESPGs of v and u, respectively.*

Lemma 4.3. *MPNN cannot distinguish one or more pairs of graphs that have non-isomorphic ESPGs.*

Figure 3 shows a pair of graphs with non-isomorphic ESPGs that BFC is able to distinguish but MPNN cannot.

4.3. DFS-guided Colouring

DFS is also a fundamental graph search method for graph problems, including detecting cycles, topological sorting, and biconnectivity. Unlike BFS whose search range grows incrementally by hop, DFS explores vertices as far as possible along each branch before backtracking. Before introducing a DFS-guided vertex colouring, recalling that we must have $v_j \prec v_i$ (or $i > j$) for any DFS back edge (v_i, v_j), we introduce two concepts used in defining η for DFS.

Definition 4.3 (Back edge crossover)**.** *Let T_v be a search tree rooted at vertex $v \in V$ and $E^{T_v}_{back}$ be the set of back edges of T_v. A relation* crossover *$\nmid$ on $E^{T_v}_{back}$ is defined as $\vec{e}_1 \nmid \vec{e}_2$, where $\vec{e}_1 = (v_{i_1}, v_{j_1})$ and $\vec{e}_2 = (v_{i_2}, v_{j_2})$, if one of the following four conditions is satisfied:*

(1) $j_2 < j_1 < i_2 < i_1$

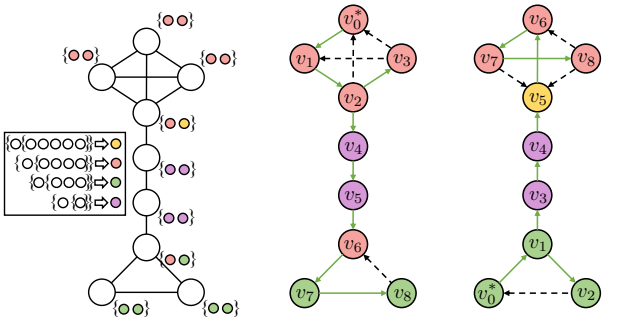


Figure 4: An uneven barbell graph and vertex colours after the first DFC iteration. 4a shows an uncoloured graph, where the vertex colours obtained from 4b and 4c are shown next to each vertex. 4b and 4c show DFC with two different roots (marked with *), where each vertex is assigned a new colour by DFC. The colour map is shown on the left. The subscripted label $v_0, \ldots, v_5$ denote the visit sequence of each vertex, e.g. v_1 is visited after v_0. For brevity, DFC is shown with only two roots.

(2) $j_1 < j_2 < i_1 < i_2$

(3) $j_1 = j_2$, $i_1 \neq i_2$

(4) $i_1 = i_2$, $j_1 \neq j_2$

It is easy to see that $\nmid$ is symmetric. That is, $\vec{e}_2 \nmid \vec{e}_1$ if and only if $\vec{e}_1 \nmid \vec{e}_2$. For instance, in Figure 4b we have $(v_2, v_0) \nmid (v_3, v_1)$. In Figure 1c, we have $(v_3, v_1) \nmid (v_5, v_2)$ and $(v_5, v_0) \nmid (v_5, v_2)$.

Definition 4.4 (Back edge cover)**.** *Let $v \in V$ be a vertex, T_v be a search tree rooted at vertex v, $\vec{e} \in E^{T_v}_{back}$ be a back edge of T_v, and $\vec{e} = (v_i, v_j)$. We say that v is* covered by *$\vec{e}$, denoted as $v \dashv \vec{e}$, if and only if $v \in \mathcal{P}^{T_v}_{v_j v_i}$, where $\mathcal{P}^{T_v}_{v_j v_i}$ is a path from v_j to v_i, formed by tree edges of T_v.*

In Figure 1c, we have $v_2 \dashv (v_3, v_1)$. In Figure 4c, we have $v_1 \dashv (v_2, v_0)$.

Depth-first colouring (DFC). Given a vertex $u \in V$, we define the set of back edges that cover vertex u as

$$Q^{T_v}_u = \{\vec{e} : u \dashv \vec{e}, \vec{e} \in E^{T_v}_{\text{back}}\} \tag{4}$$

We then find all back edges that are transitively crossovered by $Q^{T_v}_u$. We define $D^{T_v,0}_u = Q^{T_v}_u$ for the first iteration. At the k-th iteration, we define

$$\Delta D^{T_v,k}_u = \{\vec{e}_2 : \vec{e}_1 \nmid \vec{e}_2, \vec{e}_1 \in D^{T_v,k-1}_u, \vec{e}_2 \in E^{T_v}_{\text{back}}\}$$

$$D^{T_v,k}_u := \Delta D^{T_v,k}_u \cup D^{T_v,k-1}_u$$

We use $D_u^{T_v}$ to denote the set of all back edges that are transitively crossovered by edges in $Q_u^{T_v}$, i.e., when the fixed point is reached. The set of vertices covered by at least one back edge in $D_u^{T_v}$ is defined as

$$B_u^{T_v} = \{o : o \dashv \vec{e}, \vec{e} \in D_u^{T_v}, o \in V\}. \tag{5}$$

We design η for DFS, denoted η_{dfc}, as

$$\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v}) = \{o : (o, u) \in E_{\text{tree}}^{T_v}\} \cup \{o : o \in B_u^{T_v}\} \tag{6}$$

The part $\{o : (o, u) \in E_{\text{tree}}^{T_v}\}$ preserves vertices that lead to u via tree edges. $\{o : o \in B_u^{T_v}\}$ preserves vertices that are covered by the same back edges covering u, or by back edges that transitively cover u.

The vertex colouring scheme defined using Equations 1, 2, and 6 is called *depth-first colouring* (DFC). DFC is referred to as DFC-δ if the search range is limited to a δ-hop neighbourhood for each $u \in V$. Because η_{dfc} is permutation invariant, it is easy to see that DFC-δ is also permutation invariant. Figure 4 shows an example of colouring an uneven barbell graph using DFC. It can be seen that the vertices at two ends of the barbell share the same colour, while the vertices on the connecting path have different colours.

Lemma 4.4. *η_{dfc} is invariant under search order permutation.*

Distinguishing biconnectivity. We hereby show that DFC is expressive for distinguishing graphs that exhibit the biconnectivity property. Graph biconnectivity is a well-studied topic in graph theory and often discussed in the context of network flow and planar graph isomorphism (Hopcroft & Tarjan, 1973). Zhang et al. (2023) first draw attention to biconnectivity in the context of GNN and show that most GNNs cannot learn biconnectivity.

A vertex $v \in V$ is said to be a *cut vertex* (or *articulation point*) in G if removing the vertex disconnects G. Thus, the removal of a cut vertex increases the number of connected components in a graph (a connected component is an inducted subgraph of G in which each pair of vertices is connected via a path). Similarly, an edge $(v, u) \in E$ is a *cut edge* (or *bridge*) if removing (v, u) increases the number of connected components. A graph is *vertex-biconnected* if it is connected and does not have any cut vertices. Similarly, A graph is *edge-biconnected* if it is connected and does not have any cut edges.

Lemma 4.5. *Let G and H be two graphs, and $\{\!\{\lambda(u) : u \in V_G\}\!\}$ and $\{\!\{\lambda(u') : u' \in V_H\}\!\}$ be the corresponding multisets of stable vertex colours of G and H by running DFC. Then the following statements hold:*

- *For any two vertices $u \in V_G$ and $u' \in V_H$, if $\lambda(u) = \lambda(u')$, then u is a cut vertex if and only if u' is a cut vertex.*
- *For any two edges $(u_1, u_2) \in E_G$ and $(u_1', u_2') \in E_H$, if $\{\!\{\lambda(u_1), \lambda(u_2)\}\!\} = \{\!\{\lambda(u_1'), \lambda(u_2')\}\!\}$, then (u_1, u_2) is a cut edge if and only if (u_1', u_2') is a cut edge.*
- *If G is vertex/edge-biconnected but H is not, then $\{\!\{\lambda(u) : u \in V_G\}\!\} \neq \{\!\{\lambda(u') : u' \in V_H\}\!\}$.*

Corollary 4.1. *Let $u \in V_G$ and $u' \in V_H$ be two vertices, and $\lambda(u) = \lambda(u')$. Then u is in a cycle if and only if u' is in a cycle.*

4.4. Expressivity Analysis

Comparison with 1-WL. When $\delta = 1$, it is easy to see that search trees in BFC contain only direct neighbours of the root and edges between the root and its neighbours, and there are no back edges. Thus, we have the lemma below.

Lemma 4.6. *BFC-1 is equivalent to 1-WL.*

When $\delta = 1$, search trees in DFC contain direct neighbours of the root, edges between the root and the neighbours, and edges between the neighbours. We have the following lemma.

Lemma 4.7. *DFC-1 is more expressive than 1-WL.*

Corollary 4.2. *When $\delta > 1$, BFC-δ can distinguish one or more pairs of graphs that cannot be distinguished by 1-WL.*

Corollary 4.3. *When $\delta \geq 1$, DFC-δ can distinguish one or more pairs of graphs that cannot be distinguished by 1-WL.*

Taking a pair of graphs - one is two triangles and the other is one six-cycle - for example, these two graphs cannot be distinguished by 1-WL. However, they can be distinguished by BFC-2 and DFC-1 (Figure 5).

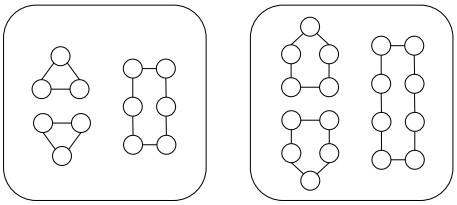


Figure 5: (Left) A graph pair can be distinguished by BFC-2 and DFC-1 but not by BFC-1 and 1-WL. (Right) A graph pair can be distinguished by BFC-3 and DFC-2 but not by BFC-2, DFC-1 and 1-WL.

Comparison with 3-WL. We can also show that, regardless of the choice of δ, BFC is no more expressive than 3-WL. This leads to the following theorem.

Theorem 4.1. *The expressive power of BFC-δ is strictly upper bounded by 3-WL.*

However, unlike BFS, DFC can distinguish graphs that cannot be distinguished by 3-WL. For example, DFC-1 can distinguish the strongly regular graph pair shown in Figure 6 which cannot be distinguished by 3-WL: the 4x4 Rook's graph of 16 vertices (Laskar & Wallis, 1999) and the Shrikhande graph (Shrikhande, 1959). In the 4x4 Rook's graph, each vertex's 1-hop subgraph has a cut vertex, while there is no cut vertex in the Shrikhande graph. So according to Lemma 4.5, DFC-1 can distinguish these two graphs. On the other hand, there are also graphs that 3-WL can distinguish but DFC-1 cannot. For example, the right graph pair in Figure 5. Thus, we have the following theorem.

Theorem 4.2. *The expressive powers of DFC-δ and 3-WL are incomparable.*

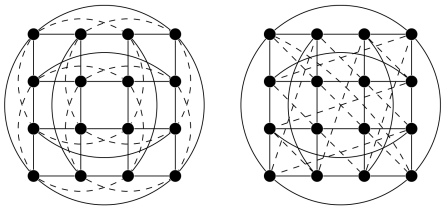


Figure 6: The 4x4 Rook's graph and the Shrikhande graph Arvind et al. (2020). Some edges are dashed for readability.

Expressivity hierarchy. The following theorem states that there exists a hierarchy among the expressive powers of BFC-δ when increasing δ.

Theorem 4.3. *BFC-δ+1 is strictly more expressive than BFC-δ in distinguishing non-isomorphic graphs.*

Theorem 4.3 implies that BFC can be used as an alternative way, separating from the WL test hierarchy, to measure the expressivity of GNNs.

Nevertheless, DFC-δ does not exhibit a hierarchy. Figure 7 depicts two pairs of non-isomorphic graph pairs: one pair can be distinguished by DFC-1 but not by DFC-2 while the other pair can be distinguished by DFC-2 but not by DFC-3. This leads to Theorem 4.4 below.

Theorem 4.4. *DFC-δ+1 is not necessarily more expressive than DFC-δ in distinguishing non-isomorphic graphs.*

5. Search Guided Graph Neural Network

We propose a graph neural network that adopts the same colouring mechanisms as BFC and DFC for feature propagation. Let $h_u^{(l)}$ be an embedding of vertex u after the l-th layer. Our search-guided graph neural network propagates embeddings over a graph via the following update rule:

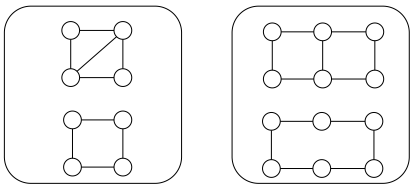


Figure 7: Non-isomorphic graph pairs. A pair can be distinguished by DFC-1 but not by DFC-2 (Left). A pair can be distinguished by DFC-2 but not by DFC-3 (Right).

$$h_u^{(l+1)} = \text{MLP}\left(\left(1+\epsilon^{(l+1)}\right)\cdot h_u^{(l)} \parallel \sum_{v\in N_\delta(u)} h_{u\leftarrow v}^{(l+1)}\right) \tag{7}$$

where

$$h_{u\leftarrow v}^{(l+1)} = \left(h_u^{(l)} + \sum_{w\in\eta_v(u)} h_{w\leftarrow v}^{(l)}\right) W_c \tag{8}$$

where, $h_u^{(0)}$ is the input vertex feature of u and $\parallel$ is the concatenation operator. $h_{w\leftarrow v}^{(l)}$ is the vertex representation of w obtained from a graph search rooted at v. When $w=v$, we have $h_{v\leftarrow v}^{(l)} = h_v^{(l)}$. Here, $W_c \in \mathbb{R}^{F\times H}$ is a learnable parameter matrix, where F and H are the dimensions of the input feature and hidden layer, respectively. Note that we use the same W_c for all vertices in $\{u \in V : c = d(u,v)\}$, which have the same shortest path distance $c = d(u,v)$ to vertex v. For brevity, we use $\eta_v(u)$ to denote $\eta(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$.

For the model to learn an injective transformation, we use a multilayer perceptron (MLP) and a learnable scalar parameter ϵ adopted from Xu et al. (2019). The model architecture closely matches the one in Equations 1 and 2, except that $\psi(\cdot)$ is replaced with a summation, $\phi(\cdot)$ is replaced by matrix multiplication, and MLP is used in place for $\rho(\cdot)$.

We term the model as *Search-guided Graph Neural Network* (SGN). When $\eta_{\text{bfc}}(\cdot)$ is used in place for $\eta_v(\cdot)$, we call it SGN-BF. When $\eta_{\text{dfc}}(\cdot)$ is used, we call it SGN-DF.

Theorem 5.1. *SGN defined by the update rule in Equations 7 and 8 and with sufficiently many layers is as powerful as LVC.*

Table 1: Time and space complexity comparison.

	MPNN	Graphormer-GD	SGN-BF	SGN-DF
Time	$\lvert V\rvert + \lvert E\rvert$	$\lvert V\rvert^2$	$\lvert V\rvert + \lvert V\rvert d^{\delta-1}$	$\lvert V\rvert + \lvert V\rvert d^{2\delta}$
Space	$\lvert V\rvert$	$\lvert V\rvert$	$\lvert V\rvert d^{\delta-1}$	$\lvert V\rvert d^{2\delta}$

Choice of δ. As of Theorem 4.3, a larger δ implies higher expressivity for SGN-BF in distinguishing isomorphic graphs. However, increasing δ also increases the size of $\eta_v(u)$, which makes it more expensive to compute as more aggregation operations are needed. Further, a larger δ means a larger receptive field at each layer, which is more likely to cause *over-squashing* (Topping et al., 2022) leading to degrade performance on vertex-level tasks, e.g. vertex classification on heterophilic graphs.

For SGN-DF, increasing δ does not necessarily increase expressivity (Theorem 4.4). However, having a larger δ means that the model can detect larger biconnected components. Therefore, in practice, we combine vertex representations from $\delta = 1$ with a larger δ for SGN-DF. This guarantees that the model is more expressive than 1-WL and allows the additional δ to be fine-tuned for each dataset and task.

Complexity. $\eta_v(u)$ in Equation 8 only needs to be computed once, along with graph searching. Therefore, the time complexity to compute $\eta_v(u)$ is on par with the adopted graph searching. Both BFS and DFS have the worst-case complexity $O(|V| + |E|)$ for searching the whole graph. Assuming d is the average vertex degree, when the search is limited to $N_\delta(v)$, we have the complexity $O(d^\delta(1 + d))$. We compute $\eta_v(u)$ for each $v \in V$ so in total it is $O(|V|d^\delta(1 + d))$. Each layer in SGN (Equation 7) aggregates $|N_\delta(u)|$ vertex representation vectors for each vertex. Each vertex representation further aggregates $|\eta_v(u)|$ vector in Equation 8, i.e., $|N_\delta(u)| \cdot |\eta_v(u)|$ operations, or, $O(d^\delta|\eta_v(u)|)$ in total. The magnitude of $|\eta_v(u)|$ can be very different for BFS and DFS, and varies from graph to graph. In the worst case of BFS where every vertex of hop $\delta + 1$ is connected to every vertex of hop δ; thus $|\eta_v(u)| = \frac{1-d^\delta}{1-d}$. In the worst case of DFS, $\eta_v(u)$ includes all vertices in $N_\delta(u)$ which has the size of d^δ. We compare the time and space complexity of our model in Table 1 with MPNN and Graphoormer-GD (Zhang et al., 2023).

6. Experiments

We evaluate SGN on two prediction tasks: vertex classification and graph classification. SGN is implemented using Pytorch and Pytorch Geometric (Fey & Lenssen, 2019). Experiments are run on a single NVIDIA RTX A6000 GPU with 48GB memory.

6.1. Vertex Classification

Datasets. We use three citation graphs, CORA, CITESEER and PUBMED, and two Amazon co-purchase graphs, COMPUTERS and PHOTO. As shown in Chien et al. (2021) these five datasets are homophilic graphs on which adjacent vertices tend to share the same label. We also use two Wikipedia graphs, CHAMELEON and SQUIRREL, and two webpage graphs, TEXAS and CORNELL, from WebKB (Pei et al., 2020). These five datasets are heterophilic datasets on which adjacent vertices tend to have different labels. Details about these datasets are shown in Table 7.

Table 2: Vertex classification results on homophilic datasets.

	COMPUTERS	PHOTO	CITESEER	CORA	PUBMED
MLP	82.9±0.4	84.7±0.3	76.6±0.9	77.0±1.0	85.9±0.2
GCN	83.3±0.3	88.3±0.7	79.9±0.7	87.1±1.0	86.7±0.3
GCN+JK†	-	-	74.5±1.8	85.8±0.9	88.4±0.5
GAT	83.3±0.4	90.9±0.7	**80.5**±0.7	88.0±0.8	87.0±0.2
APPNP	85.3±0.4	88.5±0.3	80.5±0.7	88.1±0.7	88.1±0.3
ChevNet	87.5±0.4	93.8±0.3	79.1±0.8	86.7±0.8	88.0±0.3
GPRGNN	86.9±0.3	93.9±0.3	80.1±0.8	88.6±0.7	88.5±0.3
BernNet	87.6±0.4	93.6±0.4	80.1±0.8	88.5	88.5±1.0
H_2GCN†	-	-	77.1±1.6	87.8±1.4	89.6±0.3
SGN-BF	90.7	**96.1**±0.2	78.0±1.0	88.7±0.1	**90.2**±3.5
SGN-DF	**90.9**±0.4	95.2±0.8	79.7±0.7	**89.5**±0.6	89.5±0.6

Table 3: Vertex classification results on heterophilic datasets.

	WISCONSIN	CORNELL	TEXAS	CHAMELEON	SQUIRREL
MLP	85.3±3.6	90.8±1.6	91.5±1.1	46.9±1.5	31.0±1.2
GCN	59.8±7.0	65.9±4.4	77.4±3.3	59.6±2.2	46.8±0.9
GCN+JK†	74.3±6.4	74.3±6.4	64.6±8.7	63.4±2.0	40.5±1.6
GAT	55.3±8.7	78.2±3.0	80.8±2.1	63.1±1.9	44.5±0.9
APPNP	-	**91.8**±2.0	91.0±1.6	51.8±1.8	34.7±0.6
ChevNet	82.6±4.6	83.9±2.1	86.2±2.5	59.3±1.3	40.6±0.4
GPRGNN	-	91.4±1.8	93.0±1.3	67.3±1.1	50.2±1.9
H_2GCN†	86.7±4.7	82.2±4.8	84.5±6.8	59.4±2.0	37.9±2.0
SGN-BF	**91.2**±1.0	89.5±2.7	88.7±4.3	**72.8**±0.2	**59.0**±0.3
SGN-DF	84.1±3.6	83.2±5.8	86.8±5.2	56.6±3.0	47.0±1.5

Setup and baselines. We adopt the same experimental setup as He et al. (2021), where each dataset is randomly split into train/validation/test set with the ratio of 60%/20%/20%. In total, we use 10 random splits for each dataset, and the reported results are averaged over all splits.

We compare SGN with seven baseline models: MLP, GCN (Kipf & Welling, 2017), GAT (Velickovic et al., 2017), APPNP (Klicpera et al., 2019), ChebNet (Defferrard et al., 2016), GPRGNN (Chien et al., 2021), and BernNet (He et al., 2021).

We perform a hyperparameter search on four parameters in the following ranges: number of layers $\in \{1, 2, 3, 4, 5\}$, dropout probability $\in \{0.2, 0.5, 0.7, 0.9\}$, $\delta \in \{1, 2, 3, 4\}$, and hidden layer dimension $\in \{64, 128\}$.

Observation. Results on homophilic and heterophilic datasets are presented in Tables 2 and 3, respectively. Results marked with † are obtained from Zhu et al. (2020), and other baseline results are taken from He et al. (2021).

From Tables 2 and 3, we can see that SGN generalizes to both homophilic and heterophilic graphs in vertex classifi-

cation tasks. Specifically, SGN-BF outperforms baselines in 7 out of 10 datasets, while SGN-DF outperforms 3 out of 10. SGN-BF performs better than SGN-DF in heterophilic graphs. We find that the number of vertices in $N_\delta(v)$ grows much faster for SGN-DF than SGN-BF as δ increases. This can be explained as the paths to reach each $u \in N_\delta(v)$ from v are longer in DFS than that of BFS (in BFS the path lengths are always less than or equal to δ). Therefore more vertices are included in $N_\delta(v)$ for DFS. This further implies more vertex features are aggregated for each vertex in SGN-DF, resulting in over-squashing which degrades the performance of SGN-DF on heterophilic graphs.

We also list the training runtime in Table 4. As expected, as δ increases, the runtime of SGN-BF increases but stays on par with GCN. When $\delta = 1$, SGN-DF aggregates more vertices thus is slower than SGN-BF

Table 4: Training runtime per epoch in seconds.

	GCN	SGN-BF ($\delta = 1$)	SGN-BF ($\delta = 2$)	SGN-BF ($\delta = 3$)	SGN-DF ($\delta = 1$)
CORA	0.177	0.125	0.213	0.239	0.179
PUBMED	0.349	0.224	0.315	1.271	0.219
CHAMELEON	0.205	0.198	0.457	-	0.346

6.2. Graph Classification

Datasets. We evaluate SGN on graph classification for chemical compounds, using four molecular datasets: D&D (Dobson & Doig, 2003), PROTEINS (Borgwardt et al., 2005), NCI1 (Wale et al., 2008) and ENZYMES (Schomburg et al., 2004). We also include a social dataset IMDB-BINARY. Following Errica et al. (2020), for molecular datasets, vertex features are a one-hot encoding of atom type, with the exception of ENZYMES where additional 18 features are used, whereas for IMDB-BINARY the degree of each vertex is the sole vertex feature. Details about these datasets are shown in Table 8.

Setup and baselines. We adopt the fair and reproducible evaluation setup from Errica et al. (2020), which uses an internal hold-out model selection for each of the 10-fold cross-validation stratified splits.

We compare SGN against five GNNs: DGCNN (Zhang et al., 2018), DiffPool (Ying et al., 2018), ECC (Simonovsky & Komodakis, 2017), GIN (Xu et al., 2019) and GraphSAGE (Hamilton et al., 2017), as well as two variants of the contextual graph Markov model: E-CGMM (Atzeni et al., 2021) and ICGMM (Castellana et al., 2022). We also include a competitive structure-agnostic baseline method, dubbed BASELINE, from Errica et al. (2020).

We perform the hyperparameter search: number of layers $\in \{1, 2, 3, 4, 5\}$, dropout probability $\in \{0.2, 0.5, 0.7\}$, $\delta \in \{1, 2, 3\}$, and hidden layer dimension $\in \{64\}$.

Table 5: Graph classification results.

	D&D	NCI1	PROTEINS	ENZYMES	IMDB-BINARY
BASELINE	**78.4**±4.5	69.8±2.2	75.8±3.7	65.2±6.4	70.8±5.0
DGCNN	76.6±4.3	76.4±1.7	72.9±3.5	38.9±5.7	69.2±3.0
DiffPool	75.0±3.5	76.9±1.9	73.7±3.5	59.5±5.6	68.4±3.3
ECC	72.6±4.1	76.2±1.4	72.3±3.4	29.5±8.2	67.7±2.8
GIN	75.3±2.9	80.0±1.4	73.3±4.0	59.6±4.5	71.2±3.9
GraphSAGE	72.9±2.0	76.0±1.8	73.0±4.5	58.2±6.0	68.8±4.5
E-CGMM‡	73.9±4.1	78.5±1.7	73.3±4.1	-	70.7±3.8
ICGMM‡	76.3±5.6	77.6±1.5	73.3±2.9	-	73.0±4.3
SGN-BF	76.3±3.2	78.8±2.9	74.0±3.9	64.8±7.2	71.4±7.1
SGN-DF	78.01±4.0	**81.0**±1.4	**76.1**±1.6	**66.9**±7.5	**72.3**±5.4

Observation. Results are presented in Table 5. Results marked with ‡ are obtained from Castellana et al. (2022), and other baseline results are taken from Errica et al. (2020).

We first observe that SGN-DF outperforms other GNNs and CGMM variants consistently. SGN-DF also outperforms BASELINE on 4 out of 5 datasets. Although SGN-BF also yields competitive results on several benchmarks, it does not perform better than SGN-DF. This suggests that the graph properties captured by SGN-DF, such as biconnectivity, might be useful to classify such graphs.

7. Conclusion

Inspired by the 1-WL test, we propose a new graph colouring scheme, called *local vertex colouring* (LVC). LVC iteratively refines the colours of vertices based on a graph search algorithm. LVC surpasses the expressivity limitations of 1-WL. We also prove that combining LVC with breath-first and depth-first searches can solve graph problems that cannot be solved with 1-WL test.

Based on LVC, we propose a novel variant of graph neural network, named search-guided graph neural network (SGN). SGN is permutation invariant and inherits the properties from LVC by adopting its colouring scheme to learn the embeddings of vertices. Through the experiments on a vertex classification task, we show that SGN can generalize to both homophilic and heterophilic graphs. The result of the graph classification task further verifies the efficiency of the proposed model.

Acknowledgements

We thank Professor Brendan Mckay for helpful suggestions. We acknowledge the support of the Australian Research Council under Discovery Project DP210102273. This work was also partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)) and National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1C1C1011375).

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. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2020.04.003. 7

Atzeni, D., Bacciu, D., Errica, F., and Micheli, A. Modeling edge features with deep bayesian graph networks. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pp. 1–8. IEEE, 2021. doi: 10.1109/IJCNN52387.2021.9533430. 9

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. *International Conference on Learning Representations*, 2022. 2

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021a. 1, 2

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 (ICML)*, pp. 1026–1037, 2021b. 1, 2

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1: i47–56, June 2005. 9

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. 2

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. 2, 19

Castellana, D., Errica, F., Bacciu, D., and Micheli, A. The infinite contextual graph markov model. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2721–2737. PMLR, 2022. 9

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020. 2

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In *9th International Conference on Learning Representations, ICLR*, 2021. 8

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, fourth edition*. MIT Press, April 2022. 3, 4, 17

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Neural Information Processing Systems (NeurIPS)*, pp. 3844–3852, 2016. 8

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *J. Mol. Biol.*, 330(4):771–783, July 2003. 9

Errica, F., Podda, M., Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR)*, 2020. 9

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. In *Representation Learning on Graphs and Manifolds Workshop, International Conference on Learning Representations ICLR*, 2019. 8

Geerts, F. and Reutter, J. L. Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2022. 1

Georgiev, D. and Lió, P. Neural bipartite matching. In *Workshop of the 37th International Conference on Machine Learning ICML*, 2020. 2

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 (ICML)*, pp. 1263–1272, 2017. 1

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1024–1034, 2017. 9

He, M., Wei, Z., Huang, Z., and Xu, H. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 14239–14251, 2021. 8

Hopcroft, J. and Tarjan, R. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6): 372–378, June 1973. 6, 17

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi: 10.1016/0893-6080(91)90009-T. 22

Hornik, K., Stinchcombe, M. B., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: 10.1016/0893-6080(89)90020-8. 22

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 8

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019. 8

Laskar, R. and Wallis, C. Chessboard graphs, related designs, and domination parameters. *Journal of Statistical Planning and Inference*, 76(1):285–294, 1999. ISSN 0378-3758. 7

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1

Loukas, A. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations (ICLR)*, 2020. 2

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 1, 2

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 *AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019. 1, 2, 19

Morris, C., Rattan, G., and Mutzel, P. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Neural Information Processing Systems (NeurIPS)*, 33, 2020. 1

Pei, H., Wei, B., Chang, K. C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations, ICLR*, 2020. 8

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Res.*, 32(Database issue):D431–3, January 2004. 9

Shrikhande, S. S. The Uniqueness of the L_2 Association Scheme. *The Annals of Mathematical Statistics*, 30(3): 781 – 798, 1959. 7

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 29–38. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.11. 9

Tarjan, R. E. A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2(6):160–161, April 1974. 1, 18

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 2, 8

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2017. 1, 8

Velickovic, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. Neural execution of graph algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 2, 4

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, March 2008. 9

Wang, Q., Chen, D. Z., Wijesinghe, A., Li, S., and Farhan, M. $\mathscr{N}$-WL: A new hierarchy of expressivity for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. 22

Wang, Y., Wang, Q., Koehler, H., and Lin, Y. Query-by-Sketch: Scaling shortest path graph queries on very large networks. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pp. 1946–1958, New York, NY, USA, June 2021. Association for Computing Machinery. 4

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. 2

Wijesinghe, A. and Wang, Q. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2022. 2

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019. 1, 2, 7, 9, 16, 22

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K., and Jegelka, S. What can neural networks reason about? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 2, 4

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 4805–4815, 2018. 9

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7134–7143. PMLR, 2019. 1

Zhang, B., Luo, S., Wang, L., and He, D. Rethinking the expressive power of GNNs via graph biconnectivity. In *11th International Conference on Learning Representations, ICLR 2020, Kigali, Rwanda , May 1-5, 2023*, 2023. 1, 2, 6, 8

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, 2018. 9

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations (ICLR)*, 2019. 2

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 8

A. Summary of Notations

We summarise the notations used throughout the paper in Table 6 below.

Table 6: Summary of notations.

Symbol	Description
$\{\cdot\}$	a set
$\{\!\{\cdot\}\!\}$	a multiset
$\lvert\cdot\rvert$	cardinality of a set/multiset/sequence
$\mathbb{P}(\cdot)$	a power set
G, H	undirected graphs
V	a vertex set
E	an edge set
$\vec{e}_{vu}$	directed edge from vertex v to u
(v, u)	a vertex sequence of a directed edge
$d(v, u)$	shortest distance between vertex v and u
$N_\delta(v)$	a set of vertices within δ hops of vertex v
$\mathcal{P}_{vu}$	a path from vertex v to u
$(w_0, w_1, ..., w_k)$	a vertex sequence in a path
T_v	a search tree rooted at vertex v
$v \prec u$	v precedes u in search visiting order
$v \succ u$	v succeeds u in search visiting order
$E^{T_v}_{\text{back}}$	a back edge set of the search tree T_v rooted at vertex v
$E^{T_v}_{\text{tree}}$	a tree edge set of the search tree T_v rooted at vertex v
$G \simeq H$	G and H are isomorphic
C	a set of colours
$\lambda : V \to C$	a colour refinement function that assigns a colour in C to a vertex in V
$\lambda_v : V \to C$	a colour refinement function that assigns a colour in C to a vertex in V, based on a graph search rooted at vertex v
$\eta(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})$	a function which takes a vertex u, a tree edge set $E^{T_v}_{\text{tree}}$ and a back edge set $E^{T_v}_{\text{back}}$ as input, outputs a vertex set
$\vee$	logical or
$\wedge$	logical and
$\|\|$	concatenation
$\phi(\cdot)$, $\psi(\cdot)$, $\rho(\cdot)$	injective functions
h_v	vector representation of vertex v
MLP	multilayer perceptron
ϵ	a learnable parameter
W_c	a learnable matrix with respect to the shortest path distance c between two vertices

B. Dataset Statistics

Statistics of the datasets used in our experiments are listed in Table 7 and Table 8.

C. Proofs

We first introduce several lemmas that are useful for later proofs.

Lemma C.1. *Given two pairs of vertices (u, v) and (u', v'), if $d(u, v) \neq d(u', v')$, $d(u, v) \leq \delta$, and $d(u', v') \leq \delta$, then $\lambda_v(u) \neq \lambda_{v'}(u')$ under BFC.*

Proof. We only show the case that $\lambda_v(u) \neq \lambda_{v'}(u')$ when $d(u, v) < d(u', v')$. This is because $\lambda_v(u) \neq \lambda_{v'}(u')$ can be shown for $d(u, v) > d(u', v')$ in the same way. We prove this by contradiction and thus assume that $d(u, v) = \delta'$, $d(u', v') = \delta' + \Delta$, and $\lambda_v(u) = \lambda_{v'}(u')$, where $\delta' \leq \delta$ and $\Delta \geq 1$.

Table 7: Statistics of datasets used for vertex classification.

	# Vertices	# Edges	# Features	# Classes
CORA	2708	5278	1433	7
CITESEER	3327	4552	3703	6
PUBMED	19717	44324	500	5
COMPUTERS	13752	245861	767	10
PHOTO	7650	119081	745	8
CHAMELEON	2277	31371	2325	5
SQUIRREL	5201	198353	2089	5
TEXAS	183	279	1703	5
CORNELL	183	277	1703	5
WISCONSIN	251	466	1703	5

Table 8: Statistics of datasets used for graph classification.

	# Graphs	# Classes	Avg. # vertices	Avg. # edges	# Features
D&D	1178	2	284.32	715.66	89
ENZYMES	600	6	32.63	64.14	3
NCI1	4110	2	29.78	32.30	37
PROTEINS	1113	2	39.06	72.82	3
IMDB-BINARY	1000	2	19.77	96.53	-

Since $\lambda_v(u) = \lambda_{v'}(u')$, by Equations 1 and 2, we know that $\lambda^l(u) = \lambda^l(u')$ must hold for any $l \geq \delta'$; otherwise, the injectivitity of the functions ρ and ϕ would lead to $\lambda_v(u) \neq \lambda_{v'}(u')$, contradicting with the assumption.

Then, by Equations 1 and 2 again, we have the following for any $l \geq \delta'$:

$$\psi\left(\{\!\{\lambda_v^l(w) : w \in \eta_{\text{bfc}}(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})\}\!\}\right) = \psi\left(\{\!\{\lambda_{v'}^l(w') : w' \in \eta_{\text{bfc}}(u', E^{T_{v'}}_{\text{tree}}, E^{T_{v'}}_{\text{back}})\}\!\}\right). \tag{9}$$

Because ψ is injective, the above leads to

$$|\eta_{\text{bfc}}(u, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})| = |\eta_{\text{bfc}}(u', E^{T_{v'}}_{\text{tree}}, E^{T_{v'}}_{\text{back}})| \text{ and } d(u,u) = d(u',u') = 0. \tag{10}$$

Again we know that $\lambda^{l-1}(w) = \lambda^{l-1}(w')$ because of the injectivity of ρ and ϕ. Similarly, we have the following

$$|\eta_{\text{bfc}}(w, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})| = |\eta_{\text{bfc}}(w', E^{T_{v'}}_{\text{tree}}, E^{T_{v'}}_{\text{back}})| \text{ and } d(u,w) = d(u',w') = 1. \tag{11}$$

The above can be shown recursively for all vertices in $SPG(u,v)$ and $SPG(u',v')$. So we know that, for any vertex z' in $SPG(u',v')$, there must exist a vertex z in $SPG(u,v)$ such that the following conditions hold:

$$|\eta_{\text{bfc}}(z, E^{T_v}_{\text{tree}}, E^{T_v}_{\text{back}})| = |\eta_{\text{bfc}}(z', E^{T_{v'}}_{\text{tree}}, E^{T_{v'}}_{\text{back}})| \text{ and } d(u,z) = d(u',z'). \tag{12}$$

However, since $d(u,v) = \delta'$ and $d(u',v') = \delta' + \Delta$, there must exist at least one vertex z' in $SPG(u',v')$ with $\delta' < d(u',z') \leq \delta' + \Delta$, which cannot satisfy the above conditions. This implies that $\lambda_v(u) = \lambda_{v'}(u')$ does not hold, contradicting our assumption. The proof is done.

□

Lemma C.2. $|N_\delta(v)| = |N_\delta(u)|$ *if* $\lambda(v) = \lambda(u)$.

Proof. According to Equation (1), when $\lambda(v) = \lambda(u)$, we have

$$\phi\left(\lambda(v), \psi\left\{\!\{\lambda_{w_1}(v) : w_1 \in N_\delta(v)\}\!\right\}\right) = \phi\left(\lambda(u), \psi\left\{\!\{\lambda_{w_2}(u) : w_2 \in N_\delta(u)\}\!\right\}\right).$$

Because $\phi(\cdot)$ and $\psi(\cdot)$ are injective, we have

$$\{\!\{\lambda_{w_1}(v) : w_1 \in N_\delta(v)\}\!\} = \{\!\{\lambda_{w_2}(u) : w_2 \in N_\delta(u)\}\!\}.$$

Hence, the number of elements should be the same for the multisets on the left and right sides. This leads to $|N_\delta(v)| = |N_\delta(u)|$. The proof is done. □

Lemma 4.1. *Let (u, v) and (u', v') be two pairs of vertices. Then $SPG(u, v) \simeq SPG(u', v')$ if and only if one of the following conditions hold under BFC: (1) $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$; (2) $\lambda_v(u) = \lambda_{u'}(v')$ and $\lambda_u(v) = \lambda_{v'}(u')$.*

Proof. We first show $SPG(u, v) \simeq SPG(u', v')$ if one of the two conditions holds.

We only show the statement holds when the first condition holds, i.e. "$SPG(u, v) \simeq SPG(u', v')$ if $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$". The case for the second condition, i.e. "$SPG(u, v) \simeq SPG(u', v')$ if $\lambda_v(u) = \lambda_{u'}(v')$ and $\lambda_u(v) = \lambda_{v'}(u')$", can be shown in the same way.

Assuming $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$, we show $SPG(u, v) \simeq SPG(u', v')$ by induction.

- When $d(u, v) = 0$ and $d(u', v') = 0$, we must have $u = v$ and $u' = v'$. Accordingly, both $SPG(u, v)$ and $SPG(u', v')$ contain only one node. Thus we must have $SPG(u, v) \simeq SPG(u', v')$.
- When $d(u, v) = 1$ and $d(u', v') = 1$, we have $(u, v) \in E$ and $(u', v') \in E$. Then both $SPG(u, v)$ and $SPG(u', v')$ contain only one edge. Thus we must have $SPG(u, v) \simeq SPG(u', v')$.
- When $d(u, v) = 2$ and $d(u', v') = 2$, $SPG(u, v)$ and $SPG(u', v')$ can only have one isomorphic type: assuming there are total N vertices in $SPG(u, v)$ and $SPG(u', v')$, there are $N - 2$ vertices adjacent to both u/u' and v/v'. It is easy to see $SPG(u, v) \simeq SPG(u', v')$.
- Now assume that the statement "$SPG(u, v) \simeq SPG(u', v')$ if $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$ under BFC" holds for any two pairs of vertices (u, v) and (u', v') when $d(u, v) = d(u', v') \leq \Delta$. We want to show that this statement will hold for the case $d(u, v) = d(u', v') = \Delta + 1$. It is easy to see that $\lambda_v(u) = \lambda_{v'}(u')$ if and only if $\lambda_u(v) = \lambda_{u'}(v')$, so we just need to show $SPG(u, v) \simeq SPG(u', v')$ if $\lambda_v(u) = \lambda_{v'}(u')$.

  When $d(u, v) = \Delta + 1$, we may express $SPG(u, v)$ as a tree rooted at vertex u, which has a number of children $\{\!\{SPG(u_1, v), \ldots, SPG(u_q, v)\}\!\}$ where $d(u_i, v) = \Delta$ for $1 \leq i \leq q$ and $(u, u_i) \in E$. Accordingly, we may express $SPG(u', v')$ as a tree rooted at vertex u', which has a number of children $\{\!\{SPG(u'_1, v'), \ldots, SPG(u'_q, v)\}\!\}$ where $d(u'_i, v') = \Delta$ for $1 \leq i \leq p$ and $(u', u'_i) \in E$.

  Because $\lambda_v(u) = \phi(\lambda(u), \psi(\{\!\{\lambda_v(u_1), \ldots, \lambda_v u_q\}\!\}))$ and $\lambda_{v'}(u') = \phi(\lambda(u'), \psi(\{\!\{\lambda_{v'}(u'_1), \ldots, \lambda_{v'}(u'_p)\}\!\}))$, we must have $p = q$ and $\{\!\{\lambda_v(u_1), \ldots, \lambda_v(u_q)\}\!\} = \{\!\{\lambda_{v'}(u'_1), \ldots, \lambda_{v'}(u'_p)\}\!\}$. Without loss of generality, assuming $\lambda_v(u_1) = \lambda'_v(u'_q)$, …, $\lambda_v(u_p) = \lambda'_v(u'_p)$, by our assumption for the case $d(u, v) = d(u', v') \leq \Delta$, we have $SPG(u_1, v) \simeq SPG(u'_1, v')$, …, $SPG(u_q, v) \simeq SPG(u'_p, v')$. This implies $\{\!\{SPG(u_1, v), \ldots, SPG(u_q, v)\}\!\} \simeq \{\!\{SPG(u'_1, v'), \ldots, SPG(u'_p, v')\}\!\}$. Thus, we must have $SPG(u, v) \simeq SPG(u', v')$. So the statement "$SPG(u, v) \simeq SPG(u', v')$ if $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$ under BFC" holds for the case $d(u, v) = d(u', v') = \Delta + 1$.

Now we show $SPG(u, v) \simeq SPG(u', v')$ only if one of the two conditions holds. Assuming $SPG(u, v) \simeq SPG(u', v')$, we show this by induction.

- When $d(u, v) = 0$ and $d(u', v') = 0$, we must have $u = v$ and $u' = v'$. Accordingly, both $SPG(u, v)$ and $SPG(u', v')$ contain only one node. Thus both conditions hold.
- When $d(u, v) = 1$ and $d(u', v') = 1$, we have $(u, v) \in E$ and $(u', v') \in E$. Then both $SPG(u, v)$ and $SPG(u', v')$ contain only one edge. Thus both conditions hold.
- When $d(u, v) = 2$ and $d(u', v') = 2$, $SPG(u, v)$ and $SPG(u', v')$ can only have one isomorphic type: assuming there are total N vertices in $SPG(u, v)$ and $SPG(u', v')$, there are $N - 2$ vertices adjacent to both u/u' and v/v'. It is easy to see both conditions hold.
- Now assume that the statement holds for the first condition, i.e. "$SPG(u, v) \simeq SPG(u', v')$ only if $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$ under BFC" holds for any two pairs of vertices (u, v) and (u', v') when $d(u, v) = d(u', v') \leq \Delta$. We want to show that this statement also hold for the case $d(u, v) = d(u', v') = \Delta + 1$.

  It is easy to see that $\lambda_v(u) = \lambda_{v'}(u')$ if and only if $\lambda_u(v) = \lambda_{u'}(v')$, so we just need to show $SPG(u, v) \simeq SPG(u', v')$ only if $\lambda_v(u) = \lambda_{v'}(u')$. Similar with before, when $d(u, v) = \Delta + 1$, we express $SPG(u, v)$ as a tree rooted at vertex u, which has a number of children $\{\!\{SPG(u_1, v), \ldots, SPG(u_q, v)\}\!\}$ where $d(u_i, v) = \Delta$

for $1 \leq i \leq q$ and $(u, u_i) \in E$. Accordingly, we express $SPG(u', v')$ as a tree rooted at vertex u', which has a number of children $\{\!\{SPG(u'_1, v'), \ldots, SPG(u'_q, v)\}\!\}$ where $d(u'_i, v) = \Delta$ for $1 \leq i \leq p$ and $(u', u'_i) \in E$. Because $SPG(u, v) \simeq SPG(u', v')$, we must have $p = q$. This further implies $\{\!\{SPG(u_1, v), \ldots, SPG(u_q, v)\}\!\} \simeq \{\!\{SPG(u'_1, v'), \ldots, SPG(u'_p, v')\}\!\}$. By our assumption for the case $d(u, v) = d(u', v') \leq \Delta$, we have $\{\!\{\lambda_v(u_1), \ldots, \lambda_v(u_q)\}\!\} = \{\!\{\lambda_{v'}(u'_1), \ldots, \lambda_{v'}(u'_p)\}\!\}$, which further leads to $\lambda_v(u) = \lambda_{v'}(u')$. So the statement "$SPG(u, v) \simeq SPG(u', v')$ only if $\lambda_v(u) = \lambda_{v'}(u')$ and $\lambda_u(v) = \lambda_{u'}(v')$ under BFC" holds for the case $d(u, v) = d(u', v') = \Delta + 1$.

Now assume that the statement holds for the second condition, i.e. "$SPG(u, v) \simeq SPG(u', v')$ only if $\lambda_v(u) = \lambda_{u'}(v')$ and $\lambda_u(v) = \lambda_{v'}(u')$ under BFC" holds for any two pairs of vertices (u, v) and (u', v') when $d(u, v) = d(u', v') \leq \Delta$. We can show the statement also holds for the case $d(u, v) = d(u', v') = \Delta + 1$ in the same way as before.

The proof is done. □

Lemma 4.2. *Let $v, u \in V$ be any two vertices and $\lambda(v)$ and $\lambda(u)$ be the corresponding stable colours of v and u by running BFC. We have $S_v \simeq S_u$ if and only if $\lambda(v) = \lambda(u)$, where S_v and S_u are the ESPGs of v and u, respectively.*

Proof. We first show that, if $\lambda(v) = \lambda(u)$, then we have $S_v \simeq S_u$. We prove this by induction:

- When $\delta = 0$, there is only one vertex in S_v and S_u, it is trivial to see $S_v \simeq S_u$.
- When $\delta = 1$, there are only v and its direct neighbours in S_v, and only u and its direct neighbours S_u. v and u must have the same degree for $\lambda(v) = \lambda(u)$ to hold, so we have $S_v \simeq S_u$.
- Now assume that the statement "$S_v \simeq S_u$ if $\lambda(v) = \lambda(u)$ under BFC" holds for any two vertices v and u when $\delta \leq \Delta$. We want to show that this statement will hold for the case $\delta = \Delta + 1$. When $\delta = \Delta + 1$, we only consider the case where $S_v \simeq S_u$ for $\delta = \Delta$; otherwise, we immediately have $\lambda(v) \neq \lambda(u)$ according to Lemma C.1. Assuming that there are q vertices $\{v_1, \ldots, v_q\}$ in $N_{\Delta+1}(v) \setminus N_\Delta(v)$ and p vertices $\{u_1, \ldots, u_p\}$ in $N_{\Delta+1}(u) \setminus N_\Delta(u)$, where $p \geq 1$ and $q \geq 1$. If $p \neq q$ we have $\lambda(v) \neq \lambda(u)$. So we only consider the case where $p = q$. In this case, for $\lambda(v) = \lambda(u)$ to hold, we must have $\{\!\{\lambda_{v_1}(v), \ldots, \lambda_{v_1}(v)\}\!\} = \{\!\{\lambda_{u_1}(u), \ldots, \lambda_{u_p}(u)\}\!\}$. Then we have $\{\!\{SPG(v_1, v), \ldots, SPG(v_q, v)\}\!\} \simeq \{\!\{SPG(u_1, u), \ldots, SPG(u_p, u')\}\!\}$ according to Lemma 4.1. Thus, we must have $S_v \simeq S_u$.

We then show that, if $S_v \simeq S_u$ we have $\lambda(v) = \lambda(u)$. We prove this by induction:

- When $\delta = 0$, there is only one vertex in S_v and S_u, it is trivial to see $\lambda(v) = \lambda(u)$.
- When $\delta = 1$, there are only v and its direct neighbours in S_v, and only u and its direct neighbours S_u. v and u must have the same degree for $S_v \simeq S_u$ to hold, so we have $\lambda(v) = \lambda(u)$.
- Now assume that the statement "$\lambda(v) = \lambda(u)$ if $S_v \simeq S_u$ under BFC" holds for any two vertices v and u when $\delta \leq \Delta$. We want to show that this statement will hold for the case $\delta = \Delta + 1$. Assuming that there are q vertices $\{v_1, \ldots, v_q\}$ in $N_{\Delta+1}(v) \setminus N_\Delta(v)$ and p vertices $\{u_1, \ldots, u_p\}$ in $N_{\Delta+1}(u) \setminus N_\Delta(u)$, where $p \geq 1$ and $q \geq 1$. If $p \neq q$ we have $S_v \not\simeq S_u$. So we only consider the case where $p = q$. Because $S_v \simeq S_u$ for $\delta = \Delta + 1$, we know $\{\!\{SPG(v_1, v), \ldots, SPG(v_q, v)\}\!\} \simeq \{\!\{SPG(u_1, u), \ldots, SPG(u_p, u')\}\!\}$. According to Lemma 4.1, we must have $\{\!\{\lambda_{v_1}(v), \ldots, \lambda_{v_q}(v)\}\!\} = \{\!\{\lambda_{u_1}(u), \ldots, \lambda_{u_p}(u)\}\!\}$. Thus, we have $\lambda(v) = \lambda(u)$.

The proof is done. □

Lemma 4.3. *MPNN cannot distinguish one or more pairs of graphs that have non-isomorphic ESPGs.*

Proof. Consider vertices v and u and their 1-hop neighbour vertex sets $N_1(v)$ and $N_1(u)$, respectively. We use $\lambda_{\text{WL}}(\cdot)$ to denote the colour mapping of 1-WL. We show an example where $\lambda_{\text{WL}}(v) = \lambda_{\text{WL}}(u)$ but $S_v \not\simeq S_u$. In Figure 3, S_v and S_u are non-isomorphic; however, we can see $\lambda_{\text{WL}}(v) = \lambda_{\text{WL}}(u)$ because each vertex is adjacent to the same number of vertices. We know from Xu et al. (2019) that MPNN's expressivity is upper-bounded by 1-WL; thus, the proof is done. □

Lemma 4.4. *η_{dfc} is invariant under search order permutation.*

Proof. By showing that η_{dfc} is invariant to vertex search orders, we can prove that η_{dfc} is permutation invariant.

Note that for DFS rooted at vertex v, there are multiple search trees if and only if there exist back edges. For a DFS tree T_v, any alternative DFS tree can be formed by changing a tree edge to a back edge and swap their directions. If there are no back edges in a DFS tree, the DFS tree is canonical such that the output of η_{dfc} does not change. So below we only consider the case where back edges exist.

For a non-root vertex u in G, there must be one and only one tree edge leading to u, i.e. exact one vertex in $\{o : (o, u) \in E_{\text{tree}}^{T_v}\}$. If $Q_u^{T_v}$ in Equation (4) is empty, u is not covered by a back edge and u is not in any cycle because a cycle is formed by at least one back edge (Cormen et al., 2022). Since u is not in any cycle, the vertex w precedes u in T_v is deterministic.

Now we consider the case where $Q_u^{T_v}$ is not empty, which has two further cases. Let w be a vertex preceding u in the tree edge (w, u).

- The first case is when w and u are covered together by a back edge, or by two back edges that crossover each other. In this case, u and w are in the same cycle(s), so w appears in $B_u^{T_v}$. For any alternative search tree to T_v, w and u must also be covered together by a back edge, or by two back edges that crossover each other. So the union $\{o : (o, u) \in E_{\text{tree}}^{T_v}\} \cup \{o : o \in B_u^{T_v}\}$ in Equation (6) remains the same for all possible search trees.
- The second case is when w and u are not covered by a back edge, or are covered by two back edges that do not crossover each other. In this case, (w, u) appears as a tree edge in all possible search trees rooted at v, because the tree edge (w, u) is not in any cycles. In this case, $D_u^{T_v}$ remains the same for all search trees, which makes the union $\{o : (o, u) \in E_{\text{tree}}^{T_v}\} \cup \{o : o \in B_u^{T_v}\}$ invariant to traversal order.

Thus, η_{dfc} is invariant to vertex traversal order. The proof is done. □

In the following, we introduce a lemma that is useful for proving Lemma 4.5.

Lemma C.3. *Each vertex in a biconnected component is covered by at least one DFS back edge.*

Proof. A biconnected component is an induced subgraph of G which stays connected by removing any one vertex. Therefore, each vertex in a biconnected component should participate in at least one cycle. We know a vertex forms a cycle if and only if it is covered by a DFS back edge. Hence, vertices in a biconnected component is covered by at least one DFS back edge. □

Lemma 4.5. *Let G and H be two graphs, and $\{\!\{\lambda(u) : u \in V_G\}\!\}$ and $\{\!\{\lambda(u') : u' \in V_H\}\!\}$ be the corresponding multisets of stable vertex colours of G and H by running DFC. Then the following statements hold:*

- *For any two vertices $u \in V_G$ and $u' \in V_H$, if $\lambda(u) = \lambda(u')$, then u is a cut vertex if and only if u' is a cut vertex.*
- *For any two edges $(u_1, u_2) \in E_G$ and $(u_1', u_2') \in E_H$, if $\{\!\{\lambda(u_1), \lambda(u_2)\}\!\} = \{\!\{\lambda(u_1'), \lambda(u_2')\}\!\}$, then (u_1, u_2) is a cut edge if and only if (u_1', u_2') is a cut edge.*
- *If G is vertex/edge-biconnected but H is not, then $\{\!\{\lambda(u) : u \in V_G\}\!\} \neq \{\!\{\lambda(u') : u' \in V_H\}\!\}$.*

Proof. We prove the statements in Lemma 4.5 one by one.

- By the definition that a vertex u is a cut vertex of G if removing u increases the number of connected components, Hopcroft & Tarjan (1973) shows that u is a cut vertex if one of the following two conditions is true:
  1. u is not the root of a DFS tree, and it has a child c such that no vertex in the subtree rooted with c has a back edge to one of the ancestors in the DFS tree of u.
  2. u is the root of a DFS tree, and it has at least two children.

  Equation (1) only computes vertex colour based on search trees where u is not the root, so we can just focus on condition 1. There are two types of cut vertex: one that does not form biconnected components, called the *type-1 cut vertex* (e.g. v_4 and v_5 in Figure 4b), and the other that forms biconnected components, called the *type-2 cut vertex* (v_2 and v_6 in Figure 4b). We first show that the first statement holds for the type-1 cut vertex.

For a type-1 cut vertex u, it is easy to see that we have $D_u^{T_v} = \varnothing$ because of Lemma C.3. Hence, $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ returns a single vertex set containing w, where w is the vertex preceding u in the tree edge (w, u). Note that both a type-1 cut vertex and a leaf vertex (a vertex without children) obtain a single vertex set from Equation (6); however, according to Condition 1, a leaf vertex is not a vertex tree. We show that if u is a cut vertex and x is a leaf vertex, $\lambda_G(u) \neq \lambda_G(x)$. We show this by contradiction. A leaf vertex has a degree of 1 while a cut vertex must have a degree larger than 1 (because it connects at least two biconnected components). For $\lambda_G(u) = \lambda_G(x)$ to hold, according to Equation (1), there must be the same number of vertices in $N_\delta(u)$ and $N_\delta(x)$. Because x is a leaf vertex, there is only one vertex, w_1, that contributes to the colour of u without involving other vertices. Because u is a cut vertex, there are at least two vertices, w_1' and w_2', that are adjacent to u. Assuming $\lambda_{w_1}(x) = \lambda_{w_1'}(u)$, there must exist another vertex $w_2 \in N_\delta(x)$ such that $\lambda_{w_2}(x) = \lambda_{w_2'}(u)$ and w_2 does not have an edge with x. For simplicity, we omit the superscript in Equation (2), and unpack $\lambda_{w_2}(x)$ and $\lambda_{w_2'}(u)$. We then have $\lambda_{w_2}(x) = \phi\big(\lambda(x), \psi\left(\{\!\{\lambda_{w_2}(w_1)\}\!\}\right)\big)$ and $\lambda_{w_2'}(u) = \phi\big(\lambda(u), \psi\left(\{\!\{\lambda_{w_2'}(w_2')\}\!\}\right)\big)$. Hence, we need $\lambda_{w_2}(w_1) = \lambda_{w_2'}(w_2')$. Because $\lambda_{w_2}(w_1) = \phi\big(\lambda(w_1), \psi(\{\!\{\lambda_{w_2}(w_2)\}\!\})\big)$, $\lambda_{w_2}(w_2) = \lambda(w_2)$, and $\lambda_{w_2}(w_2') = \lambda(w_2')$, we need to have $\lambda(w_2') = \phi\big(\lambda(w_1), \psi(\{\!\{\lambda_{w_2}(w_2)\}\!\})\big)$ which obviously does not hold. Therefore, if u is a cut vertex and x is a leaf vertex, $\lambda_G(u) \neq \lambda_G(x)$.

For a type-2 cut vertex u, it is obvious that $D_u^{T_v} \neq \varnothing$ and thus the colour of u is clearly different from any leaf vertices or any type-1 cut vertices. So now we just need to show $\lambda_G(u) \neq \lambda_H(x)$ when x is a non-cut vertex in a biconnected component. Because u is a type-2 cut vertex, $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ includes all vertices in the biconnected components in which u participates (at least two biconnected components), while for a non-cut vertex x, $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ includes only vertices in a single biconnected component. So a type-2 cut vertex has a different number of vertices in $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ comparing with a non-cut vertex. Therefore, it is easy to see that $\lambda_G(u) \neq \lambda_H(x)$ when u is a type-2 cut vertex and x is a non-cut vertex in a biconnected component.

Hence, if $\lambda_G(u) = \lambda_G(x)$, and x is a cut vertex if and only if u is a vertex. The proof for the first statement is done.

- According to Tarjan's algorithm for finding cut edges (Tarjan, 1974), in a DFS tree, a tree edge (u_1, u_2) is a cut edge if there is a path from u_1 to u_2 and every path from u_1 to u_2 contains edge (u_1, u_2). This can be interpreted as follows: (u_1, u_2) is a cut edge if u_1 and u_2 are not covered by the same back edge; otherwise, these back edges do not crossover each other. We prove the second statement by contradiction. If (u_1, u_2) is a cut edge and (x_1, x_2) is not a cut edge, assuming $\{\!\{\lambda(u_1), \lambda(u_2)\}\!\} = \{\!\{\lambda(x_1), \lambda(x_2)\}\!\}$, then x_1 and x_2 must be covered either by the same back edge or by different back edges that do not crossover each other. Hence, based on Equation (6), $\eta_{\text{dfc}}(x_1, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v}) = \eta_{\text{dfc}}(x_2, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$. Since (u_1, u_2) is a cut edge, we know $\eta_{\text{dfc}}(u_1, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v}) \neq \eta_{\text{dfc}}(u_2, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$. This contradicts to $\{\!\{\lambda(u_1), \lambda(u_2)\}\!\} = \{\!\{\lambda(x_1), \lambda(x_2)\}\!\}$; thus (x_1, x_2) must be a cut edge. We can swap (x_1, x_2) and (u_1, u_2) in the proof to show if (x_1, x_2) is a cut edge, (u_1, u_2) must also be a cut edge. The proof is done for the second statement.

- For a graph G to be cut vertex/edge connected, in a DFS tree, there must be a set of back edges that crossover each other and cover all vertices of G. Therefore, if $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ includes all vertices of G, then we have $\eta(u_0, E_\succ, E_\prec) = \eta(u_1, E_\succ, E_\prec) = \cdots = \eta(u_{N-1}, E_\succ, E_\prec)$ for $u_0, u_1, \ldots, u_{N-1} \in V_G$. If H is not cut vertex/edge connected, then there must exist at least one vertex $w \in V_H$ such that w is not covered by the same set of crossover back edges that cover other vertices. Thus, $\eta_{\text{dfc}}(w, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v}) \neq \eta_{\text{dfc}}(x, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ for $x \in V_H \setminus \{w\}$. We have $\{\!\{\lambda(u) : u \in V_G\}\!\} \neq \{\!\{\lambda(u') : u' \in V_H\}\!\}$ if G is vertex/edge-biconnected but H is not. The proof is done for the third statement.

□

Corollary 4.1. *Let $u \in V_G$ and $u' \in V_H$ be two vertices, and $\lambda(u) = \lambda(u')$. Then u is in a cycle if and only if u' is in a cycle.*

Proof. From the proofs of Lemmas 4.5 and C.3, we know that, if u is in a cycle, u is covered by at least one back edge. So $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ contains more than 2 vertices. Since $\lambda(u) = \lambda(u')$, $\eta_{\text{dfc}}(u', E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ also needs to contain more than two vertices, which indicates u' is in a cycle.

□

Lemma 4.6. *BFC-1 is equivalent to 1-WL.*

Proof. When $\Delta = 1$, the vertex colouring function in Equation (1) becomes

$$\lambda^{i+1}(u) := \rho(\lambda^i(u), \{\!\{\lambda_v^{i+1}(u) | (u,v) \in E\}\!\}). \tag{13}$$

Accordingly, we have Equation (2)

$$\lambda_v^{i+1}(u) := \phi\left(\lambda^i(u), \psi(\{\!\{\lambda^i(v)\}\!\})\right), \tag{14}$$

which leads to

$$\lambda^{i+1}(u) := \rho\left(\lambda^i(u), \{\!\{\phi\left(\lambda^i(u), \psi\left(\{\!\{\lambda^i(v)\}\!\}\right)\right) | (u,v) \in E\}\!\}\right). \tag{15}$$

Elements in the multiset $\{\!\{\phi\left(\lambda^i(u), \psi\left(\{\!\{\lambda^i(v)\}\!\}\right)\right) | (u,v) \in E\}\!\}$ only differ in the input of $\psi(\cdot)$. Thus, we can simplify it by defining a new injective function $\kappa(\lambda^i(v)) = \phi\left(\lambda^i(u), \psi\left(\{\!\{\lambda^i(v)\}\!\}\right)\right)$. Equation (15) becomes

$$\lambda^{i+1}(u) := \rho\left(\lambda^i(u), \{\!\{\kappa(\lambda^i(v)) | (u,v) \in E\}\!\}\right), \tag{16}$$

which is exactly the vertex colouring function of 1-WL. The proof is done. □

Lemma 4.7. *DFC-1 is more expressive than 1-WL.*

Proof. Consider two vertices v and u, and their 1-hop neighbour vertex sets $N_1(v)$ and $N_1(u)$. We use $\lambda_{\text{dfc}}(\cdot)$ and $\lambda_{\text{wl}}(\cdot)$ to denote the colour refinements by DFC-1 and 1-WL, respectively. We first show that if $\lambda_{\text{dfc}}(v) = \lambda_{\text{dfc}}(u)$, then $\lambda_{\text{wl}}(v) = \lambda_{\text{wl}}(u)$. For $\lambda_{\text{dfc}}(v) = \lambda_{\text{dfc}}(u)$ to hold, the number of vertices in $N_1(v)$ must be the same as $N_1(u)$ because of Lemma C.2. This means that v and u have the same degree, and thus $\lambda_{\text{wl}}(v) = \lambda_{\text{wl}}(u)$ must hold.

We now show that if $\lambda_{\text{wl}}(v) = \lambda_{\text{wl}}(u)$, $\lambda_{\text{dfc}}(v) = \lambda_{\text{dfc}}(u)$ may not hold. Consider the left graph pair in Figure 5, each vertex has a degree of 2 and all vertices have the same colour under 1-WL. However, the number of vertices returned by Equation (6) is not the same for the vertices in the left graph and the vertices in the right graph. Specifically, $\eta_{\text{dfc}}(u, E_{\text{tree}}^{T_v}, E_{\text{back}}^{T_v})$ returns a set of 3 vertices for each vertex in the left graph and a set of 2 vertices for the right graph. Thus, $\lambda_{\text{dfc}}(v) \neq \lambda_{\text{dfc}}(u)$. This means that DFC-1 can distinguish some pairs of non-isomorphic graphs which 1-WL cannot distinguish. The proof is done. □

Corollary 4.2. *When $\delta > 1$, BFC-δ can distinguish one or more pairs of graphs that cannot be distinguished by 1-WL.*

Proof. The left graph pair in Figure 5 can be distinguished by BFC-2 but not by 1-WL. According to Theorem 4.3, for any $\delta > 2$, BFC-δ can also distinguish this graph pair. Similarly, the right graph pair in Figure 5 can be distinguished by BFC-3 but not by 1-WL. So for any $\delta > 3$, BFC-δ can also distinguish this graph pair. □

Corollary 4.3. *When $\delta \geq 1$, DFC-δ can distinguish one or more pairs of graphs that cannot be distinguished by 1-WL.*

Proof. The left graph pair in Figure 5 can be distinguished by DFC-δ for any $\delta \geq 1$. The right graph pair in Figure 5 can be distinguished by DFC-δ for any $\delta \geq 2$. Both graph pairs cannot be distinguished by 1-WL. □

Before proving Theorem 4.1, we first define the version of the k-WL test studied by Cai et al. (1992), which is also called *folklore-WL* (FWL) (Morris et al., 2019). Let $\overrightarrow{v} = (v_1, v_2, v_3, \ldots, v_k)$ be a k-tuple. Then the neighbourhood of k-FWL is defined as a set of $n = |V|$ elements:

$$N^F(\overrightarrow{v}) = \{N_w^F(\overrightarrow{v}) | w \in V\},$$

where each $N_w^F(\overrightarrow{v})$ is defined as:

$$N_w^F(\overrightarrow{v}) = \{(w, v_2, v_3, \ldots, v_k), (v_1, w, v_3, \ldots, v_k), \ldots, (v_1, v_2, \ldots, v_{k-1}, w)\}.$$

It is known that k-WL is equivalent to (k-1)-FWL in distinguishing non-isomorphic graphs when $k > 2$ (Cai et al., 1992). Thus, below we only need to show that the expressivity of BFC is strictly upper bounded by 2-FWL.

When $k = 2$, the neighbourhood of 2-FWL is a set of n elements of a pair of 2-tuples:

$$N^F(u,v) = \{(u,w),(w,v)|w \in V\}. \tag{17}$$

Equivalently, the above equation may be written as:

$$N^F(u,v) = \{(u,w,v)|w \in V\}. \tag{18}$$

Let $G = (V, E)$ be an input graph, 2-FWL assigns colours to all pairs of vertices of G. Initially, there are three colours being assigned: *edge*, *nonedge*, and *self*. Then the colours of these pairs of vertices are refined iteratively by assigning a new colour to each pair (u, v) depending on the colours of $\{(u, w, v)|w \in V\}$ in G. This process continues until the colours of all pairs of vertices stabilize.

Let $Col(u, v)$ denote the colour of the vertex pair (u, v) by 2-FWL, and $d(u, v)$ denote the shortest-path distance between u and v. We have Lemma C.4.

Lemma C.4. *Given two pairs of vertices* (u, v) *and* (u', v')*, if* $SPG(u, v) \not\simeq SPG(u', v')$*, then* $Col(u, v) \neq Col(u', v')$.

Proof. We prove this by induction.

- When $d(u, v) = 0$ and $d(u', v') = 0$, we must have $u = v$ and $u' = v'$. Accordingly, both $SPG(u, v)$ and $SPG(u', v')$ contain only one node. Thus $SPG(u, v) \simeq SPG(u', v')$ and it is impossible to have $SPG(u, v) \not\simeq SPG(u', v')$.
- When $d(u, v) = 1$ and $d(u', v') = 1$, we have $(u, v) \in E$ and $(u', v') \in E$. Then both $SPG(u, v)$ and $SPG(u', v')$ contain only one edge. Thus $SPG(u, v) \simeq SPG(u', v')$ and it is impossible to have $SPG(u, v) \not\simeq SPG(u', v')$.
- When $d(u, v) = 2$ and $d(u', v') = 2$, we must have $(u, v) \notin E$ and $(u', v') \notin E$. By Equation 18, we also know that

$$\begin{aligned} N^F(u,v) =& \{(u,w,v)|(u,w) \in E, (w,v) \in E, w \in V\backslash\{u,v\}\}\cup \\ & \{(u,w,v)|(u,w) \in E, (w,v) \notin E, w \in V\backslash\{u,v\}\}\cup \\ & \{(u,w,v)|(u,w) \notin E, (w,v) \in E, w \in V\backslash\{u,v\}\}\cup \\ & \{(u,w,v)|(u,w) \notin E, (w,v) \notin E, w \in V\backslash\{u,v\}\}\cup \\ & \{(u,w,v)|w = u\}\cup \\ & \{(u,w,v)|w = v\} \end{aligned}$$

  Note that, each of the subsets in the above equation corresponds to a different kind of neighbour in the neighbourhood of (u, v). In this case, equivalently, $SPG(u, v) = (V_{uv}, E_{uv})$ can also be expressed as the subsets of $N^F(u, v)$ that corresponds to three kinds of neighbours in the neighbourhood of (u, v) (Lines 1, 5, and 6 in the above equation):

$$\begin{aligned} V_{uv} =& \{u,v\} \cup \{w|(u,w) \in E, (w,v) \in E, w \in V\backslash\{u,v\}\} \\ E_{uv} =& \{(u,w),(w,v)|(u,w) \in E, (w,v) \in E, w \in V\backslash\{u,v\}\} \end{aligned}$$

  We know that the colouring of 2-FWL preserves injectivity. Thus, if $SPG(u, v) \not\simeq SPG(u', v')$, it means that their corresponding subsets in $N^F(u, v)$ and $N^F(u', v')$ are not isomorphic. Then $Col(u, v) \neq Col(u', v')$.

- Now assume that the statement "if $SPG(u, v) \not\simeq SPG(u', v')$, then $Col(u, v) \neq Col(u', v')$" holds for any two pairs of vertices (u, v) and (u', v') when $d(u, v) = d(u', v') \leq \Delta$. We want to show that this statement will hold for the case $d(u, v) = d(u', v') = \Delta + 1$.

  When $d(u, v) = \Delta + 1$, we may express $SPG(u, v)$ as a tree rooted at vertex u, which has a number of children $\{SPG(u_1, v), \ldots, SPG(u_q, v)\}$ where $d(u_i, v) = \Delta$ for $1 \leq i \leq q$. Accordingly, we may express $SPG(u', v')$ in a similar way. Thus, if $SPG(u, v) \not\simeq SPG(u', v')$, there are two cases:

  (1) $\{SPG(u, u), SPG(v, v)\} \not\simeq \{SPG(u', u'), SPG(v', v')\}$
  By our assumption for the case $d(u, v) = d(u', v') \leq \Delta$, we have $\{Col(u, u), Col(v, v)\} \not\simeq \{Col(u', u'), Col(v', v')\}$. Hence, we know that $Col(u, v) \neq Col(u', v')$.

(2) $\{SPG(u,u), SPG(v,v)\} \simeq \{SPG(u',u'), SPG(v',v')\}$
Without loss of generality, we assume that $SPG(u,u) \simeq SPG(u',u')$ and $SPG(v,v) \simeq SPG(v',v')$. Then in this case $SPG(u,v) \not\simeq SPG(u',v')$ implies that $\{SPG(u_1,v),\ldots,SPG(u_q,v)\} \not\simeq \{SPG(u'_1,v'),\ldots,SPG(u'_p,v')\}$ where $(u,u_i) \in E$ and $d(u_i,v) = \Delta$ for $1 \leq i \leq q$, and $(u',u'_j) \in E$ and $d(u'_j,v') = \Delta$ for $1 \leq j \leq p$. Here, $p = q$ must hold; otherwise we immediately have $Col(u,v) \neq Col(u',v')$. By our assumption for the case $d(u,v) = d(u',v') \leq \Delta$, we have $\{Col(u_1,v),\ldots,Col(u_q,v)\} \not\simeq \{Col(u'_1,v'),\ldots,Col(u'_p,v')\}$. Thus, $Col(u,v) \neq Col(u',v')$ must hold.

□

Theorem 4.1. *The expressive power of BFC-δ is strictly upper bounded by 3-WL.*

Proof. We first seek to show 3-WL is at least as powerful as BFC-δ. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two input graphs, according to Lemma 4.2, BFC can distinguish graphs only when they have different ESPGs, e.g. $\{\!\{\lambda_{S_v}(v) : v \in V_G\}\!\} \neq \{\!\{\lambda_{S_u}(u) : u \in V_H\}\!\}$. So we just need to show $\{\!\{Col(v,v') | v' \in V_G\}\!\} \neq \{\!\{Col(u,u') | u' \in V_H\}\!\}$ for any $v \in V_G$ and $u \in V_H$ when $S_v \not\simeq S_u$. $S_v \not\simeq S_u$ implies $\{\!\{SPG(v',v) : v' \in V_G\}\!\} \not\simeq \{\!\{SPG(u',u) : u \in V_H\}\!\}$. According to Lemma C.4, we must have $\{\!\{Col(v,v') | v' \in V_G\}\!\} \neq \{\!\{Col(u,u') | u' \in V_H\}\!\}$. Thus 3-WL can distinguish graphs when they have different ESPGs, so 3-WL is at least as powerful as BFC-δ.

Now we seek to show the strictness of this bound, that is 3-WL is strictly more expressive than BFC. We show this by introducing an example graph pair in Figure 8. In this example 3-WL can distinguish the graphs but BFC cannot because the two graphs have isomorphic ESPGs. Therefore the expressiveness of BFC is strictly upper bounded by 3-WL.

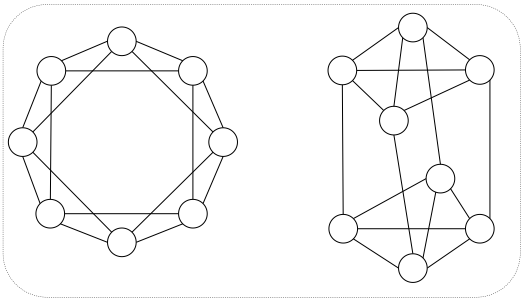


Figure 8: A pair of non-isomorphic graphs that can be distinguished by 3-WL but not by BFC.

□

To prove Theorem 4.3, we first introduce Lemma C.5.

Lemma C.5. *BFC-$\delta + 1$ is at least as expressive as BFC-δ in distinguishing non-isomorphic graphs.*

Proof. This lemma requires to show that, for any i-th iteration, if BFC-$\delta + 1$ must have the same multiset of vertex colours for G_1 and G_2, then BFC-δ must also have the same multisets of vertex colours for G_1 and G_2. Below we show for any iteration i, if the colours of any two vertices in G_1 and G_2 are the same by BFC-$\delta + 1$, then their colours by BFC-δ must also be the same. We show this by induction.

- For $i = 0$, it is obvious that the initial vertex colours are the same for BFC-$\delta + 1$ and BFC-δ.
- For $i > 0$, we assume that this statement "if $\lambda^i(u_1) = \lambda^i(u_2)$ for BFC-$\delta + 1$ then $\lambda^i(u_1) = \lambda^i(u_2)$ for BFC-δ" holds for the i-th iteration, and seek to show that the statement also holds for the $(i+1)$-th iteration. We show this by contradiction. Assuming $\lambda^{i+1}(u_1) = \lambda^{i+1}(u_2)$ hold for BFC-$\delta + 1$ but not for BFC-δ, we have

$$\rho(\lambda^i(u_1), \{\!\{\lambda^i_v(u_1) | v \in N_{\delta+1}(u_1)\}\!\}) = \rho(\lambda^i(u_2), \{\!\{\lambda^i_v(u_2) | v \in N_{\delta+1}(u_2)\}\!\}) \tag{19}$$

and

$$\rho(\lambda^i(u_1), \{\!\{\lambda^i_v(u_1) | v \in N_{\delta}(u_1)\}\!\}) \neq \rho(\lambda^i(u_2), \{\!\{\lambda^i_v(u_2) | v \in N_{\delta}(u_2)\}\!\}). \tag{20}$$

Because $\lambda^{i+1}(u_1) = \lambda^{i+1}(u_2)$ for BFC-$\delta+1$, we must have $\lambda^i(u_1) = \lambda^i(u_2)$ for BFC-$\delta+1$. According to our assumption "if $\lambda^i(u_1) = \lambda^i(u_2)$ for BFC-$\delta+1$ then $\lambda^i(u_1) = \lambda^i(u_2)$ for BFC-δ", we have $\lambda^i(u_1) = \lambda^i(u_2)$ hold for BFC-δ. Thus, we can simplify Equations (19) and (20) as

$$\{\!\{\lambda_v^i(u_1)|v \in N_{\delta+1}(u_1)\}\!\} = \{\!\{\lambda_v^i(u_2)|v \in N_{\delta+1}(u_2)\}\!\} \tag{21}$$

$$\{\!\{\lambda_v^i(u_1)|v \in N_{\delta}(u_1)\}\!\} \neq \{\!\{\lambda_v^i(u_2)|v \in N_{\delta}(u_2)\}\!\} \tag{22}$$

Because $N_\delta(u) \subseteq N_{\delta+1}(u)$ and $N_{\delta+1}(u) - N_\delta(u) = \{v|d(u,v) = \delta+1\}$, there must exist at least one pair of vertices u' and u'', where $d(v,u') = \delta+1$ and $d(v,u'') \leq \delta$, such that $\lambda_v^i(u') = \lambda_v^i(u'')$. This contradicts Lemma C.1. So the assumption "$\lambda^{i+1}(u_1) \neq \lambda^{i+1}(u_2)$ for BFC-δ" must not hold. Thus, the statement "if $\lambda^{i+1}(u_1) = \lambda^{i+1}(u_2)$ for BFC-$\delta+1$ then $\lambda^{i+1}(u_1) = \lambda^{i+1}(u_2)$ for BFC-δ" holds.

This means that, for any iteration i, if the colours of any two vertices in G_1 and G_2 are the same by BFC-$\delta+1$, then their colours by BFC-δ must also be the same. The proof is done. □

Now we prove Theorem 4.3.

Theorem 4.3. *BFC-δ+1 is strictly more expressive than BFC-δ in distinguishing non-isomorphic graphs.*

Proof. By Lemma C.5, we know that BFC-$\delta+1$ is at least as expressive as BFC-δ. Now we just need to show that there exists at least one pair of non-isomorphic graphs $(\hat{G}_1, \hat{G}_2)$ that can be distinguished by BFC-$\delta+1$ but not by BFC-δ.

Inspired by Wang et al. (2023), we hereby show a specific construction of such graph pairs using cycles. We construct $\hat{G}_1$ to be two cycles of length $2\delta+1$, and $\hat{G}_2$ to be one cycle of length $4\delta+2$, for any $\delta \geq 1$. $\hat{G}_1$ and $\hat{G}_2$ can be distinguished by BFC-$\delta+1$ but not by BFC-δ. Figure 5 shows two examples graph pairs constructed using this method. The proof is done. □

Theorem 5.1. *SGN defined by the update rule in Equations 7 and 8 and with sufficiently many layers is as powerful as LVC.*

Proof. The concatenation operator in Equation 7 preserves injectivity. So Equation 7 is equivalent of replacing $\rho(\cdot)$ in Equation 1 with an MLP. The summation operator in Equation 8 is the injective variant of $\psi(\cdot)$ in Equation 2. The multiplication with W_c is a single-layer MLP that used in place for $\phi(\cdot)$ in Equation 2. As a universal approximator (Hornik, 1991; Hornik et al., 1989), MLP can learn injective functions. Hence, SGN can be shown to be as expressive as LVC following the proof of Theorem 3 by Xu et al. (2019). We leave the details out for brevity. □