Hyperset Individualisation for the Graph Isomorphism Problem

Hyperset Individualisation for the Graph Isomorphism Problem

Simone Boscaratto[1,*,†], Francesco Nascimben[1,*,†] and Alberto Policriti[1,*,†]

[1]Dipartimento di Scienze Matematiche, Informatiche e Fisiche, Università degli Studi di Udine, Via delle Scienze, 206, Udine, 33100, Italy

Abstract

In this work, we propose a novel approach for graph canonisation called *Hyperset Individualisation* (HI), combining node-individualisation and bisimulation reduction on a set-theoretic framework in an effort to tackle the Graph Isomorphism problem on simple graphs. Building on this idea, we define a variant of HI which enhances its expressiveness by integrating the structural information provided by *points* (PHI); moreover, we define and study two non-equivalent k-dimensional versions of the algorithm (HIE_k and HIA_k). Finally, we present an iterative refinement framework based on rendering of node partitions by means of atoms.

Keywords

Graph isomorphism, graph canonisation, hypersets, bisimulation, individualisation

1. Introduction

The long-standing Graph Isomorphism complexity problem, i.e. the problem of determining the complexity of establishing whether two graphs are isomorphic or not, remains open to this day.

Particularly relevant to it is the thoroughly studied Weisfeiler-Leman algorithm. Originally introduced in its 2-dimensional version [1] and later generalised by Babai and Mathon [2], the k-dimensional Weisfeiler-Leman algorithm WL_k produces a stable colouring of the k-tuples of nodes $\mathbf{v} \in V^k$ of an input graph $G = \langle V, E \rangle$. Basically, it produces a *one-way error* multiset $\mathsf{WL}_k(G)$ containing the stable colours of all tuples: two isomorphic graphs always result in the same multiset, while the converse does not hold true in general. Hence, it provides an incomplete, although statistically very accurate [3], test for graph isomorphism: larger values of k correspond to larger classes of correctly identified graphs [4], at the cost of increasing time complexity. To this day, *colour refinement* (WL_1) remains the most used algorithm to practically identify graphs up to isomorphism [5].

The first aim of this paper is to introduce and study a *Hyperset Individualisation* algorithm HI, in an attempt to give an alternative view to tackle the Graph Isomorphism problem with respect to WL_1. In this novel approach, *node individualisation* and *bisimulation reduction* techniques are combined in a set-theoretic framework to produce a certificate for each given undirected simple graph: such a certificate can be computed in time $\mathcal{O}(nm \log n)$, while two certificates can be compared in time $\mathcal{O}(n^2 m \log n)$, where n is the number of nodes and m the number of edges of the given graphs. Although node individualisation is a well-known concept in the field of graph canonisation (see, e.g., [6, 7]), and so is bisimulation reduction in several other fields (see, e.g., [8, 9]), to the best of our knowledge the proposed algorithm is the first attempt at using the two of them together. As HI is shown to be incomparable with WL_1, we will focus on hard instances for the latter, showing how HI can be more effective on them while keeping the same time complexity. When HI itself is not sufficient to break down these cases, we improve it in three different ways: PHI (*Pointed* Hyperset Individualisation) and the "k-dimensional" versions of HI, namely HIE_k and HIA_k (Hyperset Individualisation *of order k* using the *Empty set* or

ICTCS 2025: Italian Conference on Theoretical Computer Science, September 10–12, 2025, Pescara, Italy

*Corresponding author.

†These authors contributed equally.

simone.boscaratto@uniud.it (S. Boscaratto); francesco.nascimben@uniud.it (F. Nascimben); alberto.policriti@uniud.it (A. Policriti)

0009-0008-8192-6898 (S. Boscaratto); 0009-0009-2863-165X (F. Nascimben); 0000-0001-8502-5896 (A. Policriti)

Atoms, respectively).

Finally, the proposed methods are enriched by iteratively refining an initial partition of the node set using HI. This is implemented by interleaving two steps. In the first, a finer, ordered partition of the nodes is produced by computing bisimulation on every "anchored" version of the input graph. In the second, the partition computed in the preceding step is *rendered* by orderly adding new atoms—encoded by nodes of a suitable gadget graph.

This paper is organised as follows: in Section 2, we introduce the adopted notation and the tackled problem, recalling some essential notions. In Section 3 we define the (basic) Hyperset Individualisation algorithm, showing its accomplishments and failures. On these grounds, Section 4 shows the improvements that can be put forward by focusing on the aforementioned versions of HI. In Section 5 we describe how encoding an initial partition on the node set—while keeping a strict set-theoretic perspective—can enhance the expressiveness of our method; furthermore, we show how to apply the latter in an iterative fashion in order to progressively refine said initial partition. In Section 6 we discuss implementations and provide some analysis of the computational complexity of all our methods. Finally, in Section 7 we draw conclusions and state some open problems, mainly concerning the expressiveness of the aforementioned methods.

2. Basics

In this paper, standard graph-theoretic and set-theoretic notations will be adopted. In particular, a tuple will be delimited by angled parentheses $\langle\ \rangle$ and a multiset by the parentheses $\{\!\{\ \}\!\}$; unordered pairs will be represented as $\{\cdot\,,\cdot\}$, as they often do in existing literature. The symbol $\uplus$ will denote the multiset *sum* operation, which sums the multiplicities of each element, common or not, of the addend multisets. Connections between two nodes will be referred to as *edges* in the case of undirected graphs and as *arcs* in the directed case; $G = \langle V_G, E_G\rangle$ (resp., $\vec{G} = \langle V_G, \vec{E}_G\rangle$) will denote an undirected (resp., directed) graph, while subscripts will be omitted whenever clear from the context and unless otherwise specified. In this case, by n and m we will denote, respectively, the number of nodes and edges (or arcs) of a graph. Finally, $N_G(v) = \{w : \{v, w\} \in E_G\}$ (resp., $N^+_G(v) = \{w : \langle v, w\rangle\} \in \vec{E}_G\}$) denotes the set of *neighbours* (resp., *out-neighbours*) of the node v in an undirected graph G (resp., a directed graph $\vec{G}$).

We will mainly treat the case of finite undirected graphs (without self-loops and weighted or multiple edges), also referred to as *simple* graphs. At due time, we will also require these graphs to be connected.

Consider then a pair of simple graphs $G = \langle V_G, E_G\rangle$ and $H = \langle V_H, E_H\rangle$.

Definition 2.1. An *isomorphism* between two simple graphs G and H is a bijection $\pi\colon V_G \to V_H$ which preserves both *adjacencies* and *non-adjacencies*, i.e. $\{u, w\} \in E_G \Leftrightarrow \{\pi(u), \pi(w)\} \in E_H$ for all nodes u, w in V_G. If G and H are graphs labelled on nodes (in particular, coloured graphs), it is intended that an isomorphism also has to preserve the nodes' labels. If such π exists, G is said to be *isomorphic* to H, denoted by $G \cong H$.

Given two graphs G and H, the *Graph Isomorphism problem* (from now on, also abbreviated as *GI*) consists in establishing whether $G \cong H$ or not.

2.1. Weisfeiler-Leman Algorithm

The 1-dimensional Weisfeiler-Leman (WL_1, for short), also known as *colour refinement*, is the simplest algorithm of the WL family. Given a graph $G = \langle V, E\rangle$, WL_1 produces a *stable* colouring of its nodes, in a sequence of at most $n - 1$ iterations. Let C^1_i be the colouring produced by WL_1 after its i-th iteration: since we only consider uncoloured graphs as input, we assume the initial colouring C^1_0 to be uniform for all nodes. At step $i \geq 1$ and for each node $v \in V$, WL_1 collects the colours of v's neighbours into a multiset $M_i(v)$, called the *aggregation map* of v at step i: a new colour $C^1_i(v)$ is then computed from

(and uniquely associated to) v's previous colour and its current aggregation map, by means of a perfect hash function HASH. M_0 is not defined.

$$M_i(v) \coloneqq \left\{\!\!\left\{ C^1_{i-1}(w) : w \in N(v) \right\}\!\!\right\} \qquad C_i(v) \coloneqq \textsc{hash}\Big(C^1_{i-1}(v), M_i(v)\Big)$$

Two nodes share the same colour at step i if and only if they shared the same colour at step $i-1$ *and* their aggregation maps match. This refinement procedure repeats until no colour class can be further split, i.e. until a stable colouring C^1_∞ is reached: termination is guaranteed by the finiteness of V. We define $\mathsf{WL}_1(G) \coloneqq \{\!\!\{C^1_\infty(v) : v \in V\}\!\!\}$.

The generalised k-dimensional Weisfeiler-Leman algorithm (WL_k, for short) produces a stable colouring of the k-tuples in V^k. Each $\mathbf{v} \in V^k$ is initially coloured with its *atomic type* $\mathrm{atp}(\mathbf{v})$, describing the (ordered) subgraph induced on G by this tuple. Formally, two k-tuples $\mathbf{u} = \langle u_1, .., u_k\rangle$ and $\mathbf{w} = \langle w_1, .., w_k\rangle$ have the same atomic type if and only if the mapping $u_i \mapsto w_i$ is an isomorphism from the G-subgraph induced by $\mathbf{u}$ to the G-subgraph induced by $\mathbf{w}$. We denote the initial colour of each tuple $\mathbf{v}$ by $C^k_0(\mathbf{v}) \coloneqq \mathrm{atp}(\mathbf{v})$. As for the 1-dimensional version, WL_k also proceeds by repeated aggregation of neighbouring colours: given $\mathbf{v} = \langle v_1, .., v_k\rangle$ and a node w, the tuple obtained by replacing exactly one of its nodes v_i with w is the i-th w-neighbour of $\mathbf{v}$, which we denote by $\mathbf{v}_{i,w}$. The colour-update schema is similar to the one described for WL_1, the sole difference being that the aggregation map of a tuple $\mathbf{v}$ is now a multiset of *k-tuples* of colours, one for each node w in G.

$$M^k_i(\mathbf{v}) \coloneqq \left\{\!\!\left\{ \langle C^k_{i-1}(\mathbf{v}_{1,w}), \ldots, C^k_{i-1}(\mathbf{v}_{k,w})\rangle : w \in V \right\}\!\!\right\} \qquad C^k_i(\mathbf{v}) \coloneqq \textsc{hash}\Big(C^k_{i-1}(\mathbf{v}), M^k_i(\mathbf{v})\Big)$$

Again, the refinement procedure iterates until a stable colouring C^k_∞ is reached. We define $\mathsf{WL}_k(G) \coloneqq \{\!\!\{C^k_\infty(\mathbf{v}) : \mathbf{v} \in V^k\}\!\!\}$; WL_k can be implemented to run in time $\mathcal{O}(n^{k+1}\log n)$ on n-vertex graphs.

2.2. Graph theory elements

In this paper, a few common notions of graph theory will be mentioned and thus used along the path leading to the definition and refinement of our algorithms. We will recall the relevant definitions here.

Definition 2.2 (Regular graph)**.** A graph $G = \langle V, E\rangle$ is said to be *d-regular* if all nodes $v \in V$ have the same degree $\deg(v) = d$.

Definition 2.3 (Strongly regular graph)**.** A graph $G = \langle V, E\rangle$ is said to be *strongly regular* (*SRG*s for short) with parameters $\langle n, d, \lambda, \mu\rangle \in \mathbb{N}^4$ if:

- $|V| = n$;
- $\forall v \in V,\ \deg(v) = d$, i.e. G is d-regular;
- $\forall \{u, w\} \in E,\ |N(u) \cap N(w)| = \lambda$, i.e. all adjacent nodes have λ-many common neighbours;
- $\forall \{u, w\} \notin E,\ |N(u) \cap N(w)| = \mu$, i.e. all non-adjacent nodes have μ-many common neighbours.

Regular and, in particular, strongly regular graphs will be taken into account as notoriously they are hard instances for WL_1 and WL_2 (see Section 3).

Definition 2.4 (Automorphism)**.** An *automorphism* on a simple graph $G = \langle V, E\rangle$ is a bijection $\pi\colon V \to V$ such that $\pi(G) = G$, i.e. π preserves the adjacencies and non-adjacencies of G. π is a *non-trivial* automorphism if π is not the identity function on V. If G is a a graph labelled on nodes, it is intended that an automorphism also has to preserve the nodes' labels.

Definition 2.5 (Orbit partition)**.** Let $G = \langle V, E\rangle$ be a graph, let 2^V be the powerset of its nodes. The (node) *orbit partition* of G is the coarsest partition $O \subset 2^V$ of V such that all nodes belonging to the same class can be mapped into each other by an automorphism: such classes are called the (node) *orbits* of G.

Definition 2.6 (Rigid (asymmetric) graph)**.** A graph $G = \langle V, E \rangle$ is said to be *rigid* (or *asymmetric*) if its orbit partition is *discrete*, i.e. $|O| = n$, and so G only admits the identity on V as an automorphism.

Definition 2.7 (Vertex-transitive graph)**.** A graph $G = \langle V, E \rangle$ is said to be *vertex-transitive* if its orbit partition is *unit*, i.e. $|O| = 1$, and so given any pair of nodes $u, w \in V$, there is an automorphism mapping u into w.

Remark 1. A vertex-transitive graph is regular. In particular, every complete graph (clique) K_n and every cycle C_n are vertex-transitive.

As a useful sample for testing our proposed algorithms, we define the classic Cai-Fürer-Immermann (CFI) construction as presented in [4] and list its main features.

The relevance of CFI graphs lies in their ability to bound the expressiveness of Weisfeiler-Leman algorithms: indeed, for any given $k > 2$, it is possible to build a non-isomorphic graph pair distinguished by WL_{k+1}, but not by WL_k; as $\{\mathsf{WL}_k : k \in \mathbb{N}^+\}$ is a sequence of algorithms with increasing expressive power, CFI graphs draw the line between what can and what cannot be told at a given step of the hierarchy.

Before delving into this graph-building technique, we require a preliminary definition.

Definition 2.8 (Separator)**.** Given a connected graph $G = \langle V, E \rangle$, a *separator* of G is a subset $S \subset V$ of nodes whose removal causes the induced subgraph on $V \setminus S$ to have no connected components with cardinality greater than $|V|/2$.

Given a connected graph G whose vertices have all degree at least two, its CFI graph $X(G)$ is built as follows. Each vertex with degree d is expanded in a graph X_d with the following features.

- Nodes: for each $i \in \{1, \ldots, d\}$ there are two nodes denoted by a_i, b_i; for each subset $S \subseteq \{1, \ldots, d\}$ with *even* cardinality there is a node M_S representing it, as well.
- Edges: there is an edge between a_i and M_S if and only if $i \in S$; there is an edge between b_i and M_S if and only if $i \notin S$.

X_d has 2^{d-1} nodes and the same number of colour-preserving automorphisms (assuming that for each i there is a colour c_i labelling a_i and b_i and no other) by [4, Lemma 6.1]. Given a vertex v such that $\deg(v) = d$, call $X(v)$ the graph X_d associated to v. Then, for each neighbour w of v choose an $i \in \{1, \ldots, d\}$ and rename a_i and b_i as $a(v, w)$ and $b(v, w)$, respectively. $X(G)$ is constructed by adding edges $\{a(v, w), a(w, v)\}$ and $\{b(v, w), b(w, v)\}$ for each $\{v, w\} \in E_G$. Finally, given $\{v, w\} \in E$, a single *twist* in $X(G)$ replaces edges $\{a(v, w), a(w, v)\}$ and $\{b(v, w), b(w, v)\}$ with $\{a(v, w), b(w, v)\}$ and $\{b(v, w), a(w, v)\}$, thus building a "twisted" graph $\tilde{X}(G)$.

It is proven [4, Lemma 6.2] that graphs built through this method are either isomorphic to $X(G)$ (if the number of twists is even) or to $\tilde{X}(G)$ (otherwise), while $X(G) \ncong \tilde{X}(G)$. [4, Theorems 5.2, 6.4] prove that, given any graph G whose smallest separator has cardinality at least $s + 2$, the resulting CFI graphs $X(G)$ and $\tilde{X}(G)$ cannot be distinguished by WL_s, for $s \geq 1$. Knowing that graphs with arbitrary minimum separator size can be constructed [10], this guarantees that WL_k cannot be a complete isomorphism test, for any $k \in \mathbb{N}$.

2.3. Hypersets and Bisimulation

The set-theoretic framework for our algorithms is here introduced.

A *well-founded* set x is such that every descending chain in the membership relation starting from it is finite and acyclic, meaning that $x \ni x_1 \ni \cdots \ni x_n$ eventually halts for a finite n—in a *pure* set theory, x_n is always the empty set $\emptyset$—and $x \neq x_i \neq x_j$ for every pair $i \neq j, i, j \in \{1, \ldots, n\}$. To compare two well-founded sets, the *extensionality* criterion is applied: two sets are equal if and only if they have the same elements.[1]

[1] This is very common to many standard set theories, in particular Zermelo-Fraenkel's; see, e.g., [11].

On the contrary, a *non-well-founded* set, or *hyperset*, admits loops in the membership relation: with such a move we grant the existence of "extra" sets x such that, for instance, $x \in x$, or $x \in x_1 \in \cdots \in x_n \in x$. By overcoming the limits imposed by a well-founded definition of $\in$, Forti and Honsell [12], and then Aczel [13], created a richer universe in which (hyper-)sets like $\Omega = \{\Omega\}$ are admitted. However, by allowing membership loops, we also allow multiple alternative representations of the same hyperset (e.g., Ω is the hyperset solving the set-theoretic equations $\zeta = \{\zeta\}$, $\zeta = \{\{\zeta\}\}$, and so on). This can be seen more easily by translating the set-theoretic representations of a hyperset into their graph-theoretic equivalents, on which we will rely upon to define a proper equality criterion for non-well-founded sets.

We refer to [13] and [14] for the following definitions. Consider a directed graph $\vec{G} = \langle V, \vec{E} \rangle$ with finitely many nodes and no labelled or multiple edges between them; loops and self-loops are admitted. By choosing a distinguished node $p \in V$, we get the *pointed graph* $\langle V, \vec{E}, p \rangle$, with p as its point. Moreover, if we can draw a path from p to every other node in the graph, $\langle V, \vec{E}, p \rangle$ is called *accessible pointed graph*, or *apg* for short. A *decoration* of an apg is an assignment of sets as labels to its nodes in such a way that the children of a node are labelled as the elements of the node's label. By a slight abuse of notation, an arc $\langle u, w \rangle \in \vec{E}$ is then interpreted as the relationship $w \in u$. The *anti-foundation axiom* (AFA) by Aczel simply states that each apg admits a unique decoration.

Moving forward, we need to understand whether two apgs represent the same hyperset, i.e. if their points are labelled with the same hyperset according to AFA. To handle this, the following definition is needed to outline an equality criterion compatible with hypersets.

Definition 2.9 (Bisimulation, bisimilarity)**.** Let $\vec{G} = \langle V, \vec{E} \rangle$ be a directed graph. A binary relation $\flat$ among the nodes of V is said to be a *bisimulation* on $\vec{G}$ if $u \flat w$ with $u, w \in V$ always implies that:

- for every child u' of u there exists a child w' of w such that $u' \flat w'$, and
- for every child w' of w there exists a child u' of u such that $u' \flat w'$.

It is easy to see that when every node in a directed graph has out-degree at least one, the universal relation is a bisimulation.

The union of two bisimulations is itself a bisimulation and the union of all bisimulations on $\vec{G}$ is called *bisimilarity*. All bisimulations are reflexive and symmetric relations; the following result shows that bisimilarity is also transitive, and thus it is an equivalence relation on the set of its nodes [13].

Theorem 2.1. *Bisimilarity on a directed graph $\vec{G} = \langle V, \vec{E} \rangle$ is the coarsest bisimulation on $\vec{G}$ and it defines an equivalence relation on the set of its nodes, denoted by $\equiv_{\vec{G}}$.*

The notion of bisimilarity constitutes an equality criterion for hypersets: stating that two apgs (i.e., their points) are bisimilar means that they both represent the same hyperset. Intuitively, one can think of bisimilarity as the coarsest partition of the nodes set grouping functionally equivalent ones: assuming $u \equiv_{\vec{G}} w$, any "move" performed starting from u can be mirrored starting from w. Notice that the existence of an isomorphism π between two oriented graphs implies that every vertex v of the first graph is bisimilar to its image $\pi(v)$; the converse does not hold true in general.

Remark 2. Consider that colour refinement computes the *graded bisimulation* on a graph. "Simple" and graded bisimulation differ as the former is able to express a sentence of the form "there is *at least* one arc connecting a node in class A to a node in class B" (hence proving itself useful for automata minimisation and modal logic), while the latter can express "there are *exactly* k arcs connecting a node in class A to a node in class B" (thus it is used with counting logic). For more about this topic, see [15].

Define the *transitive closure* $\mathrm{trCl}(x)$ of a (hyper-)set x as the set containing its elements, and their elements, and so on. More formally:

$$\mathrm{trCl}(x) \coloneqq x \cup \bigcup_{y \in x} \mathrm{trCl}(y)$$

If an apg can be decorated by associating to each node either the set corresponding to its point, or a distinct element of its transitive closure, it will be referred to as the *pointed membership graph* of the (hyper-)set labelling its point. If a hyperset h has finite transitive closure, then we can represent it by a finite pointed membership graph: in this case we will say that h is a *hereditarily finite rational hyperset*.[2]

Notice also that, if an apg describes the hyperset h, then either the apg coincides with its pointed membership graph, or at least two of its nodes are bisimilar: in the latter case, we can *collapse* the bisimilar nodes into each other, i.e. replace each class of nodes from the partition given by bisimilarity with just one node, and replace all the arcs from a class to another with an arc between the resulting nodes (see also [8]). In this way, we are always able to reduce the given apg exactly to the pointed membership graph of the hyperset dubbing its point, which is its representation with the fewer number of nodes: we would then say that the pointed membership graph of h is the *(maximum) bisimulation contraction* of the given apg.

We now consider two useful definitions to separate non-bisimilar nodes. The first one is the concept of *rank*: for a well-founded (i.e. acyclic) apg, the rank of each node is simply the maximum length of a path issuing from it, but for a non-well-founded one giving the right definition is a much more delicate issue. Since in this paper we will heavily use the tool provided by [8], we will also rely upon the same source for our definition.

Definition 2.10. Let $\vec{G} = \langle V, \vec{E} \rangle$ be a directed graph, then define $\vec{G}^{scc} \coloneqq \langle V^{scc}, \vec{E}^{scc} \rangle$ where

$$V^{scc} \coloneqq \{c : c \text{ is a strongly connected component in } G\}$$
$$\vec{E}^{scc} \coloneqq \{\langle c_1, c_2 \rangle : (c_1 \neq c_2) \wedge (\exists v_1 \in c_1)(\exists v_2 \in c_2)(\langle v_1, v_2 \rangle \in \vec{E})\}$$

Notice that for any $\vec{G}$, the corresponding $\vec{G}^{scc}$ is always acyclic, by definition of strongly connected components.

Definition 2.11 (Rank)**.** Let $\vec{G} = \langle V, \vec{E} \rangle$ be a directed graph and $\vec{G}^{scc} = \langle V^{scc}, \vec{E}^{scc} \rangle$ be the graph of its strongly connected components; let $c \colon V \to V^{scc}$ be the function mapping each node in V to its strongly connected component. The *rank* of v is defined as

$$\mathrm{rk}(v) \coloneqq \begin{cases} 0 & \text{if } v \text{ is a leaf in } G \\ -\infty & \text{if } c(v) \text{ is a leaf in } \vec{G}^{scc} \\ & \text{and } v \text{ is not a leaf in } \vec{G} \\ \max\Big(\big\{1 + \mathrm{rk}(w) : \langle c(v), c(w) \rangle \in \vec{E}^{scc}, w \in \mathrm{WF}(\vec{G})\big\} & \\ \qquad \cup \big\{\mathrm{rk}(w) : \langle c(v), c(w) \rangle \in \vec{E}^{scc}, w \notin \mathrm{WF}(\vec{G})\big\}\Big) & \text{otherwise} \end{cases}$$

where $\mathrm{WF}(\vec{G})$ is the well-founded (loop-free) part of $\vec{G}$.

This definition of rank is by no means unique: as long as the restriction to the well-founded case matches the standard definition, the other details may vary.

The *distance* between two nodes can be regarded as the dual to the previous definition, since it is defined as the length of the *shortest* path instead of the longest. In particular, we will be interested in the distance $\mathrm{dist}(v, v_\emptyset)$ from a node v to a node representing the empty set—i.e. a node with only incoming arcs. Such distance will be finite for every node of the directed graphs processed by any of our proposed algorithms, as we shall make clear in the following sections.

Notice how nodes at different ranks or different distances from the empty set cannot be in the same bisimilarity class.

[2] To see that every hyperset can have a graphical representation with infinitely many nodes, it is sufficient to "unwrap" its loops (e.g., Ω can be represented by an infinitely descending chain, see [13]); to match this definition, though, it is sufficient that there exist finite ones. Set-theoretically, a hereditarily finite rational hyperset is defined as the solution to a finite system of finite set equations, see e.g. [14].

2.4. Node Individualisation

Individualisation of single nodes, for the purpose of symmetry breaking, is a well-known technique in the field of *graph canonisation*, i.e. the problem of efficiently computing a certificate (e.g. a code, a colouring, an ordering of the nodes) that uniquely identifies any given graph up to isomorphism. Most state-of-the-art practical tools, such as `nauty` and `Traces` [7], rely on the so called *individualisation-refinement* paradigm, which alternately applies WL_1 to the graph and assigns a unique colour to one node from a non-singleton class (chosen according to some heuristic) until a discrete node partition is reached. Although this approach generates a (potentially) exponentially large tree of colourings, appropriate heuristics and exploitation of discovered automorphisms allow such tools to prune significant parts of the tree, leading to fast performances in most cases [7].

Individualisation finds applications in important theoretic results. t-CR bounded graphs, defined in [16] as those graphs for which a discrete colouring can be obtained by repeatedly applying WL_1, selecting a non-singleton class of size at most t and assigning a unique colour to each of its nodes, play a key role in Grohe et al.'s test for isomorphism running in time $n^{\text{polylog}(h)}$ for n-vertex graphs excluding some h-vertex graph as a minor [17].

In both previous scenarios, this simmetry-breaking technique is iteratively paired with WL_1 in order to reach a discrete colouring. Besides, in the HI framework introduced below, individualisation is applied only once for each node v in the graph, before launching a bisimulation algorithm which produces a hyperset h_v associated to v.

3. The Hyperset Individualisation Algorithm

The first definition of the *Hyperset Individualisation* algorithm HI is aimed at giving a set-theoretic cut to the graph canonisation problem by assigning a *multiset of hypersets* to each undirected simple graph. As for WL_1, the result obtained after performing HI is a multiset encoding pieces of information about the given graph; however, while these are based on node degrees for the former, the latter computes bisimilarity contractions by interpreting the graph as the pointed membership graph of some hyperset.

Due to the following lemma, we can, and will, restrict our analysis to *connected* simple graphs.

Lemma 3.1. *Let $G = \langle V_G, E_G\rangle$ and $H = \langle V_H, E_H\rangle$ be undirected, possibly non-connected simple graphs. Let $G' = \langle V_G \cup \{s_G\}, E_G \cup \{\{s_G, v\} : v \in V_G\}\rangle$ and $H' = \langle V_H \cup \{s_H\}, E_G \cup \{\{s_H, v\} : v \in V_H\}\rangle$ be the graphs obtained by adding to both a source node reaching each node of the original graphs. Then, $G \cong H$ if and only if $G' \cong H'$.*

HI pseudocode is reported as Algorithm 1. After replacing each (undirected) edge of $G = \langle V, E\rangle$ with a pair of opposite arcs, thus resulting in its "bi-oriented" version $\vec{G}$, a new arc from a node $v \in V$ to a new node $v_\emptyset$—which, from a set-theoretic perspective, represents the empty set—is added. Given the so-modified pointed graph $\vec{G}_v$, with v itself as the point, a tool such as the one defined in [8] can be applied to get its bisimulation contraction, resulting in the pointed membership graph of a hyperset h_v. Then, (the pointed membership graph of) h_v is added to a multiset and the operation is repeated for every node of the original graph: the resulting multiset is the certificate given by the algorithm to the input graph.[3]

We will say that the directed graph $\vec{G}_v$ is *anchored* in v, or that v is the *anchor* of $\vec{G}_v$ (the currently *individualised* node). Whenever the underlying graph G is clear, the notation $\equiv_v$ will be used to denote the (maximum) bisimulation equivalence between nodes of $\vec{G}_v$.

Remark 3. If bisimulation contractions were performed on a connected simple graph without the individualisation of any node by the empty set, the unique apg in the resulting multiset would be either the one of the empty set $\emptyset$ itself or the one of the hyperset Ω. Thus, it would only distinguish the graph having just one node (and no edges) from any other graph.

[3] Formally, the algorithm presented in [8] outputs a representation of a graph, so the correct picture for the certificate $\mathsf{HI}(G)$ is a multiset of pointed membership graphs; however, we will often say that it is a multiset of the hypersets themselves.

Algorithm 1 Hyperset individualisation HI

Require: $G = \langle V, E \rangle$ ▷ Undirected, simple, connected
Ensure: $\mathsf{HI}(G)$
1: $\vec{E} \leftarrow \emptyset$
2: **for** $\{u, w\} \in E$ **do**
3: $\quad \vec{E} \leftarrow \vec{E} \cup \{\langle u, w \rangle, \langle w, u \rangle\}$ ▷ Replace an edge with a pair of arcs
4: **end for**
5: $\mathsf{HI}(G) \leftarrow \{\!\!\{\ \}\!\!\}$ ▷ Initialise $\mathsf{HI}(G)$ as the empty multiset
6: **for** $v \in V$ **do**
7: $\quad \vec{G}_v \leftarrow \langle V \cup \{v_\emptyset\}, \vec{E} \cup \{\langle v, v_\emptyset \rangle\}, v \rangle$ ▷ Append a node labelled as the empty set $\emptyset$ to v
8: $\quad h_v \leftarrow \mathsf{DPP}(\vec{G}_v)$ ▷ Run the bisimulation algorithm defined in [8] on $\vec{G}_v$
9: $\quad \mathsf{HI}(G) \leftarrow \mathsf{HI}(G) \uplus \{\!\!\{h_v\}\!\!\}$ ▷ Add the resulting hyperset to $\mathsf{HI}(G)$
10: **end for**

Remark 4. For any $\vec{G}_v$, the only node of rank 1 is v itself; furthermore, by definition of bisimulation, two nodes of different rank will not collapse in h_v. The same considerations apply to the distance from the empty set.

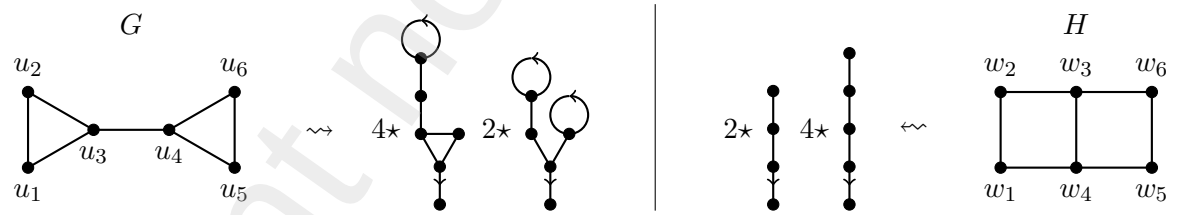
The following result shows that HI is a one-way error test for GI, much as WL_1 is.

Lemma 3.2. *Let G, H be a pair of connected simple graphs. Then, $\mathsf{HI}(G) \neq \mathsf{HI}(H)$ implies $G \ncong H$.*

Proof. By contradiction, assume $G \cong H$, then there exists an isomorphism $\pi \colon G \to H$ between them. Given any pair of nodes $u \in G$ and $w = \pi(u) \in H$, $\vec{G}_u \cong \vec{H}_w$ by an isomorphism $\tilde{\pi} = \pi \cup \langle u_\emptyset, w_\emptyset \rangle$, where $u_\emptyset, w_\emptyset$ denote the nodes representing the empty set in $\vec{G}_u$ and $\vec{H}_w$, respectively. In particular, the bisimulation contractions $h_u \in \mathsf{HI}(G)$ and $h_w \in \mathsf{HI}(H)$ are such that $h_u = h_w$. By applying this reasoning to each pair of isomorphic nodes, we get a contradiction. □

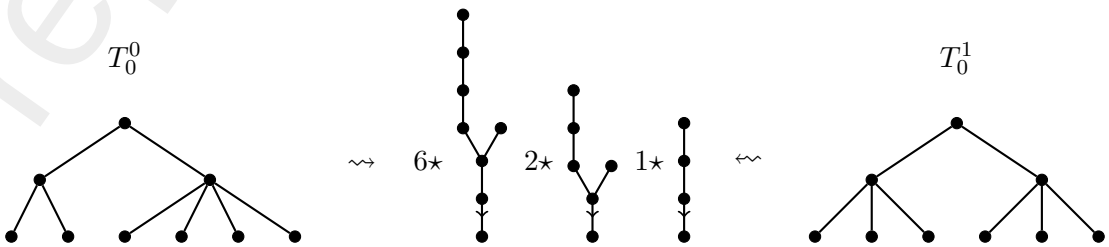
From now on, we will say that two nodes u, w are HI-equivalent if $h_u = h_v$. For what concerns expressiveness of HI, consider the following two examples.

Example 3.1. HI can distinguish non-isomorphic graph pairs which WL_1 cannot. Indeed, by performing HI on the graphs G and H below, we obtain the multisets containing the depicted hypersets with the reported multiplicities (recall that each undirected edge represents a pair of opposed arcs in hypersets).



Observe that in $\mathsf{HI}(G)$ (resp., $\mathsf{HI}(H)$) there are four copies of the hyperset obtained by individualising u_i (resp., w_i) with $i \in \{1, 2, 5, 6\}$ and two copies of the one obtained by individualising u_i (resp., w_i) with $i \in \{3, 4\}$. As they differ between the two graphs, HI distinguishes G and H; on the contrary, WL_1 assigns the same colour to nodes with matching index in both graphs.

Example 3.2. WL_1 can identify non-isomorphic trees, but the following tree pair is not distinguished by HI. Indeed, for both graphs, nodes at the same depth yield the same hyperset after individualisation.



As the previous examples show, the expressive powers of HI and WL_1 are incomparable. Notice that both of the presented graph pairs are distinguished by WL_2.

In the following sections we will show the first achievements of our algorithm, particularly focusing on regular graph: by making counting ability (Weisfeiler-Leman's strength) irrelevant, we isolate HI's behaviour on these hard instances of GI.

3.1. Polynomial-time isomorphism check cases

First and foremost, we point out two scenarios in which HI allows to check for isomorphism in polynomial time.

Lemma 3.3. *Let the connected simple graphs $G = \langle V_G, E_G \rangle$, $H = \langle V_H, E_H \rangle$ be such that $|V_G| = |V_H| = n$ and $\exists h \in \mathsf{HI}(G) \cap \mathsf{HI}(H) : |\mathrm{trCl}(h)| = n$. Then $G \cong H$.*

Proof. Notice that the h of the statement has maximum transitive closure: there are no bisimulation collapses among the n nodes originally in G or H. Any two graphs $\vec{G}_u$, $\vec{H}_w$ associated to h in $\mathsf{HI}(G)$ and $\mathsf{HI}(H)$ are, except in the connection to the empty set $\emptyset$, isomorphic to G and H themselves, respectively. The corresponding discrete partition induced by h on both G and H naturally yields an isomorphism between the two. □

Lemma 3.4. *Let the connected simple graphs $G = \langle V_G, E_G \rangle$, $H = \langle V_H, E_H \rangle$ be such that $|V_G| = |V_H| = n$ and $\mathsf{HI}(G) = \mathsf{HI}(H) \in \mathrm{Set}$, i.e. they contain the same collection of n pairwise distinct hypersets. Then, both graphs have discrete orbit partitions, and the bijection $\pi : V_G \to V_H$ such that $\pi(u) = w$ if and only if $h_u = h_w$ is the only possible isomorphism between G and H.*

Proof. Two nodes producing different hypersets cannot belong to the same orbit (if they are mapped into each other by an automorphism, then they are bisimilar, thus they produce the same hyperset once individualised). Therefore, only bijections exchanging nodes whose individualisation produces the same hyperset are candidate isomorphisms. □

If the conditions of Lemma 3.3 hold, then $G \cong H$ can be declared directly at runtime (check Section 6), as soon as the same hyperset h of maximum transitive closure is found on both graphs. If the assumptions of Lemma 3.4 hold, instead, the (single) candidate isomorphism induced by the sets of n distinct hypersets can be checked in time $\mathcal{O}(n + m)$. In light of this, pinpointing classes of graphs for which either of the two lemmas always apply would prove extremely useful.

We observe that the assumptions of the two lemmas on the given graphs are not equivalent. Indeed, consider a connected simple graph G:

1. $(\exists h \in \mathsf{HI}(G) : |\mathrm{trCl}(h)| = n) \not\Rightarrow \mathsf{HI}(G) \in \mathrm{Set}$: if $G = L_n$ is a line graph with length $n > 1$, by appending the empty set to one of its extrema we obtain a hyperset with maximum transitive closure, yet $\mathsf{HI}(L_n)$ has all its hypersets in pairs, except at most one;

2. $\mathsf{HI}(G) \in \mathrm{Set} \not\Rightarrow (\exists h \in \mathsf{HI}(G) : |\mathrm{trCl}(h)| = n)$: if G is the (rigid, 3-regular) Frucht graph (Figure 1), $\mathsf{HI}(G)$ is a set, yet one of the produced hypersets has no maximum transitive closure.

One might be tempted to assume that the discrete orbits of rigid graphs would yield a discrete identification of nodes through bisimulation, either at the single hyperset- or at the overall HI-certificate-level. However, Figure 2 shows an example of a rigid graph such that its HI-certificate is a set, but *none* of its hypersets has maximum transitive closure, thus violating the premises of Lemma 3.3; besides, the HI-certificate of the rigid graph of Figure 3 contains n copies of the same hyperset, thus violating Lemma 3.4. We conclude that, even on rigid graphs, HI is not guaranteed to define a single, decisive candidate isomorphism or, in other terms, a complete canonical order on the nodes.

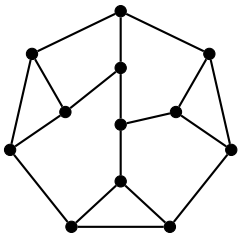


Figure 1: The Frucht graph. It is the first smallest 3-regular rigid graph to be found (see [18]). One can verify by hand that, by appending the empty set to each node, the resulting hypersets have maximum transitive closure—thus, by rigidity, they all come in single copies—except for the one on the top, in which case the transitive closure reduces to 6 nodes.

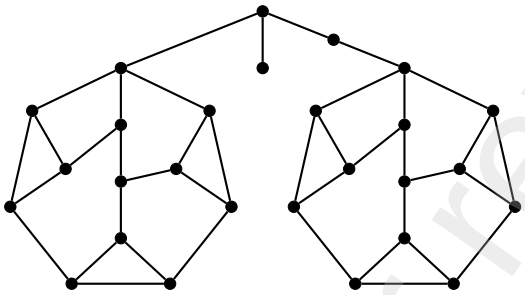


Figure 2: A rigid graph built with two copies of the Frucht graph; the different length of the chains connecting them ensures rigidity is kept. By appending the empty set to any node of one of the copies—except for its highest point in the picture—the corresponding copy do not get any collapse, but the other one does. Despite this, no multiple copies of the same pointed membership graph appear in its HI-certificate.

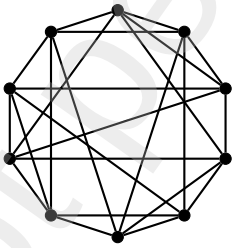


Figure 3: An asymmetric 5-regular graph with 10 nodes, whose existence is guaranteed by [19] (one can verify with `nauty` that there are exactly two of them). Each node, once individualised, produces the same hyperset after maximum bisimulation contraction, with the same number of nodes collapsed in each bisimilarity class.

3.2. Strongly regular graphs

Consider Figure 4: the depicted pair of strongly regular graphs is distinguished by WL_3, but by none among WL_1, WL_2, and HI: as known, WL_1 does not distinguish them by simple regularity; moreover, WL_2 cannot tell apart any pair of SRGs with identical parameters [21]. By testing HI on them, we obtain that each individualisation on those graphs yields the same hyperset, thus producing the same certificate.

The following lemma proves that this is not an isolated case, by providing a complete characterisation of the possible $\mathsf{HI}(G)$ that can be obtained when G is a strongly regular graph. Before stating and proving it, we need a more specific classification of strongly regular graphs (see [22]).

Definition 3.1 (Imprimitive and primitive strongly regular graphs)**.** Let G be a strongly regular graph. Then, G is called *imprimitive* if G or its complement $\bar{G}$ are disconnected, and *primitive* otherwise.

As shown in [22], a SRG identified by parameters $\langle n, d, \lambda, \mu\rangle$ is imprimitive if and only if $\mu \in \{0, d\}$ and thus $\lambda \in \{d-1, 0\}$, respectively.

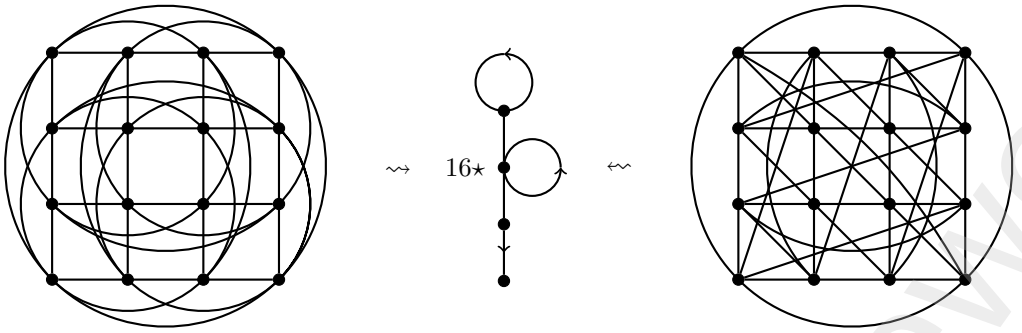


Figure 4: The 4×4 Rook's graph (left) and the Shrikhande graph (right), as pictured in [20]. These graphs are both strongly regular with the same parameters $\langle n, d, \lambda, \mu \rangle = \langle 16, 6, 2, 2 \rangle$ (see Definition 2.3). However, they are not isomorphic as the neighbours of a node form two separate 3-cycles in the Rook's graph, while they form a single 6-cycle in the Shrikhande graph. HI produces for the both of them a multiset containing 16 copies of the same hyperset (centre).

Lemma 3.5 (HI-certificates for strongly regular graphs)**.** *Let G be a connected strongly regular graph with parameters $\langle n, d, \lambda, \mu \rangle$ and consider Figure 5. The following cases occur.*

1. *If G is imprimitive, then:*
   a) *if $\mu = 0$, $\mathsf{HI}(G)$ has n copies of the hyperset (a),*
   b) *if $\mu = d$ and $n = 2d$, $\mathsf{HI}(G)$ has n copies of the hyperset (b),*
   c) *if $\mu = d$ and $n > 2d$, $\mathsf{HI}(G)$ has n copies of the hyperset (c).*
2. *If G is primitive, then:*
   c) *if $\lambda = 0$, $\mathsf{HI}(G)$ has n copies of the hyperset (d)*
   d) *if $\lambda > 0$, $\mathsf{HI}(G)$ has n copies of the hyperset (e).*

Proof. We refer to [22, Chapter 9] for the following proof. First and foremost, notice how every (connected component of a) strongly regular graph has diameter at most 2: thus, when individualisation of a single node and bisimulation contraction are performed, the maximum distance from the empty set is 3. Besides, all the nodes at the same distance from any anchor v are bisimilar in $\vec{G}_v$, as a trivial consequence of strong regularity: once v is individualised, every node at the same distance from it must have (the same number of) neighbours in each class.

Case 1.a. If $\mu = 0$, then G is a disjoint union of cliques; in particular, G is connected only if $G = K_n$. Chosen any v as anchor, all the other nodes have an arc to v and between them, then h_v reduces to hyperset (a) (from the collapse of $V \setminus \{v\}$ a self-loop is generated).

Case 1.b. If $\mu = d$ and $n = 2d$, then G is the complete bipartite graph $K_{2\times d}$, where both anticliques have d nodes each. Chosen any $v \in V$ as anchor, $N(v)$ collapses in a single node at distance 2 from $v_\emptyset$ without a self-loop, since $N(v)$ is a single anticlique and therefore there are no edges within it; the $d - 1$ nodes left, i.e. those nodes belonging to the same anticlique as v, collapse without a self-loop at distance 3 for the same reason.

Case 1.c. If $\mu = d$ and $n > 2d$, then G is the complete *multipartite* graph $K_{a\times k}$, composed by a-many anticliques of k nodes each, with $n = ak$, $d = (a-1)k = \mu$, $a > 2$. Chosen any $v \in V$ as anchor, nodes collapse as in the previous case: however, the resulting single node at distance 2 from $v_\emptyset$ has a self-loop, since $N(v)$ is composed by multiple anticliques fully connected to each other.

Case 2.d. If $\lambda = 0$, then G is triangle-free. Chosen any $v \in V$ as anchor and $w \in N(v)$, v and w have no common neighbours: $N(v)$ thus collapses in a single node without self-loop at distance 2 from $v_\emptyset$. Then, take $w' \in V$ at distance 3 from $v_\emptyset$: v and w' are such that $|N(v) \cap N(w')| = \mu \in \{1 \dots, d-1\}$, hence the other $d - \mu > 0$ neighbours of w' are at the same distance from $v_\emptyset$ as w' itself, thus producing a self-loop when all the nodes at that distance collapse.

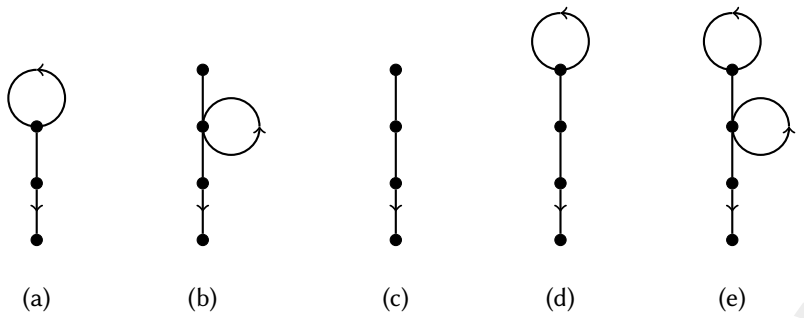


Figure 5: The five possible hypersets produced by HI as applied to strongly regular graphs (Lemma 3.5).

Case 2.e. By a similar reasoning to the one of Case 2.d, we get a self-loop at distance 3 from $v_\emptyset$.[4] Moreover, as any anchor v and a $w \in N(v)$ form a triangle, the nodes at distance 2 from $v_\emptyset$ collapse with a self-loop. □

Corollary 3.1. HI *cannot tell apart SRGs with identical parameters.*

3.3. Tools

After applying HI on (strongly) regular graphs, we need to consider the general case. In order to handle it, we introduce the notion of *gadget graph*, which can be used to encode properties of nodes (such as their degrees) or any arbitrary partition, thereby enhancing the expressiveness of our methods, while maintaining a strict set-theoretic view. This tool will turn out to be useful along the rest of the paper.

Definition 3.2 (Gadget graph)**.** For each connected simple graph $G = \langle V, E\rangle$ with $|V| = n$, its *gadget graph* $\vec{N}_G = \langle V_{N_G}, \vec{E}_{N_G}\rangle$ with $|V_{N_G}| = 2n + 1$ is defined as:

$$\vec{N}_G \coloneqq \Big\langle \big\{v_{\{\emptyset\}^k}, v_{\overline{k}} : k \in \{0,\dots,n\}\big\}, \big\{\langle v_{\{\emptyset\}^k}, v_{\{\emptyset\}^{k-1}}\rangle, \langle v_{\overline{k}}, v_{\overline{k-1}}\rangle, \dots, \langle v_{\overline{k}}, v_{\overline{0}}\rangle : k \in \{1,\dots,n\}\big\}\Big\rangle,$$

where $\{\emptyset\}^0 = \emptyset = \overline{0}$, $\{\emptyset\}^k = \{\{\emptyset\}^{k-1}\}$ and $\overline{k} = \{\overline{0},\dots,\overline{k-1}\}$. Nodes $v_{\{\emptyset\}^k}$ encode the Zermelo ordinals (super-singletons of the empty set), while nodes $v_{\overline{k}}$ encode the von Neumann ordinals. Any such node is called an *atom.*

From a practical standpoint, a gadget graph with n non-bisimilar nodes is sufficient, as it allows us to encode a discrete partition to a n-nodes graph (basically, it enumerates them). However, two distinct sets of ordinals make the discussion simpler as it allows a straightforward correspondence between them and the nature of encoded information (see Section 5).

As none of the atoms can occur in any individualisation of $\vec{G}$ due to its characteristic bi-orientation, they can be used to enforce non-bisimilarity between arbitrary nodes $u, w \in \vec{G}$, simply by adding arcs $\langle u, a_1\rangle, \langle w, a_2\rangle$ pointing to distinct atoms $a_1 \neq a_2$.

4. Improvements on HI

In this section, we will focus on different techniques to improve our algorithm in order to get a finer partition of the universe of all graphs.

The first version (PHI) focuses on rotating a hyperset's point across all nodes in order to increase expressiveness, while the second (HIE$_k$) and the third (HIA$_k$) methods, in addition, address a different question: is it possible to define a hierarchy of HI-like algorithms, in a format similar to the one of the WL$_k$ family?

[4] Observe that this common feature is summed up in [22, Proposition 9.3.1].

4.1. Pointed Hyperset Individualisation

A new point of view named *Pointed Hyperset Individualisation* (PHI), will be here defined. Given a connected simple graph G, the certificate $\mathsf{PHI}(G)$ can be not only a multiset of hypersets, like $\mathsf{HI}(G)$, but a (richer) matrix of hypersets encoding G more explicitly than the adjacency matrix itself.

Before defining the matrix $\mathsf{PHI}(G)$, observe that, as long as the graph is finite, we can always define a pre-order on the nodes of any connected simple graph.

Definition 4.1 (Rank pre-order)**.** Let $G = \langle V, E\rangle$ be a connected simple graph and consider $\mathsf{HI}(G) = \{\!\{h_v : v \in V\}\!\}$. For each $v \in V$, define a list ℓ_v where the ranks of all nodes within $\vec{G}_v$ are sorted in increasing order. Given $u, w \in V$, we would write $u \preceq w$ if and only if $\ell_u \leq_L \ell_w$, where $\leq_L$ is the lexicographical ordering of the lists.

The aforementioned lists can be computed by slightly modifying the algorithm proposed in [8], at the same computational cost of HI (check Section 6 for a more detailed description of this approach). For what concerns a global ordering of pointed membership graphs, some attempts have been made e.g. in [23] and [24].

Next, observe that we can define a variant of the hypersets appearing in the HI-certificate where the point is not necessarily the anchor itself, but it can be a different (bisimilarity class of a) node. This will increase the expressiveness of our original method, as we shall state in Lemma 4.2, after laying down some necessary definitions.

Definition 4.2. Let $G = \langle V, E\rangle$ be a connected simple graph and $v, w \in V$. Then, define the hyperset h_v^w, *anchored in* v *and pointed in* w, as the maximum bisimulation contraction of the graph $\vec{G}_v^w := \langle V \cup \{v_\emptyset\}, \vec{E} \cup \{\langle v, v_\emptyset\rangle\}, [w]_{\equiv_v}\rangle$.

Basically, h_v^w is "structurally" identical to the previously defined h_v, i.e. their membership graphs coincide, as they are both obtained from anchoring v. However, (the $\vec{G}_v$-bisimilarity class of) w is dubbed as the point of the newly defined h_v^w: in this way, when h_v^w is given, we are able to retrieve the bisimilarity class of the node w w.r.t. the individualisation of v.

We are now ready to present the matrix generalising HI.

Definition 4.3 (Pointed Hyperset Individualisation matrix)**.** Let $G = \langle V, E\rangle$ be a connected simple graph with $|V| = n$ nodes, and consider its certificate $\mathsf{HI}(G)$. For each $h_v \in \mathsf{HI}(G)$, and for every $w \in V$, let h_v^w be as in Definition 4.2. Given an enumeration $v_1, \ldots, v_n$ of the nodes in G which is compatible with the pre-order of Definition 4.1, define $\mathsf{PHI}(G)$ as

$$\mathsf{PHI}(G) := \begin{pmatrix} h_{v_1}^{v_1} & \cdots & h_{v_1}^{v_j} & \cdots & h_{v_1}^{v_n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{v_i}^{v_1} & \cdots & h_{v_i}^{v_j} & \cdots & h_{v_i}^{v_n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{v_n}^{v_1} & \cdots & h_{v_n}^{v_j} & \cdots & h_{v_n}^{v_n} \end{pmatrix}.$$

An obvious issue when introducing such a matrix is: how can we order the nodes of a graph so that isomorphic graphs get identical matrices? In other words, for now we have just re-stated the graph isomorphism problem as expressed by means of adjacency matrices. Let A_G and A_H be the adjacency matrices of the simple graphs G and H, respectively; a *permutation matrix* is an invertible square matrix $P \in \{0, 1\}^{n\times n}$ having exactly one entry equal to 1 for each row and for each column, whose inverse coincides with its transposed matrix P^T. Then, GI can be re-stated as:

Does there exist a permutation matrix P such that $A_G = P^T A_H P$?

Similarly, without an canonical ordering of the nodes of a graph, we are now asking:

Does there exist a permutation matrix P such that $\mathsf{PHI}(G) = P^T \mathsf{PHI}(H) P$?

Although any answer to the former implies the same answer to the latter (and vice versa, as we prove below), PHI encodes strictly more information than the adjacency matrix does, thus allowing us to sensibly reduce the search space for candidate permutation matrices. Firstly, as we explicitly ask that the ordering given by the lists of ranks is kept, we can exclude matrices swapping nodes with the same list; secondly, for instance, we can ignore the ones swapping nodes whose rows or columns contain different apgs (or the same apgs, but in different numbers) in $\mathsf{PHI}(G)$.

The existence of an ordering such that $\mathsf{PHI}(G) = \mathsf{PHI}(H)$ is equivalent to $G \cong H$, as proven below.

Theorem 4.1. *Let G and H be connected simple graphs with n nodes and consider a bijective map $\pi\colon G \to H$ such that $\pi(u_i) = w_i$ for all $i \in \{1, \dots, n\}, u_i \in V_G, w_i \in V_H$. Then, π is an isomorphism between G and H if and only $\mathsf{PHI}(G) = \mathsf{PHI}(H)$.*

Proof. If π is an isomorphism, $\mathsf{PHI}(G) = \mathsf{PHI}(H)$ is trivial. Moreover, if $\mathsf{PHI}(G) = \mathsf{PHI}(H)$ then $\forall i, j \in \{1, \dots, n\}, h_{u_i}^{u_j} = h_{w_i}^{w_j}$. Therefore, $\{u_i, u_j\} \in E_G$ if and only if $\{w_i, w_j\} \in E_H$: otherwise, the relative hypersets would be distinct. This completes the proof. □

Remark 5. Notice that, according to Definition 4.2, each row $\hbar_{v_i} = \langle h_{v_i}^{v_1}, \dots, h_{v_i}^{v_n} \rangle$ consists of hypersets with the same structure as h_{v_i}, but with the point varying over each node in V: this allows us to count the number of nodes collapsed into each class of h_{v_i}, an information which is dismissed by base HI. Each column $\hbar^{v_j}$, instead, highlights the same node v_j as the point of the pointed membership graphs obtained by varying the anchor over each node in V, i.e. it collects all the set-theoretic "functional roles" played by v_j over all anchorings. Observe also that $\mathsf{HI}(G) = \{\!\{h_{v_i}^{v_i} : i \in \{1, \dots, n\}\}\!\}$.

It should be noted that two rows $\hbar_{v_i}, \hbar_{v_j}$ of $\mathsf{PHI}(G)$ can never be equal, as the pointed hypersets at column i necessarily mismatch (as we observed, the anchor v_i is always the only node at rank 1): the same holds at column j. Similarly, two columns $\hbar^{v_i}, \hbar^{v_j}$ of $\mathsf{PHI}(G)$ can never be equal due to mismatches at rows i and j.

For this reason, we introduce the following definitions.

Definition 4.4 (Row-equivalent nodes)**.** We say that $v_i, v_j \in V$ are *row-equivalent* if and only if their rows $\hbar_{v_i}, \hbar_{v_j}$ contain the same unordered elements, i.e. $\{\!\{h_{v_i}^{w} : w \in V\}\!\} = \{\!\{h_{v_j}^{w} : w \in V\}\!\}$.

Definition 4.5 (Column-equivalent nodes)**.** We say that $v_i, v_j \in V$ are *column-equivalent* if and only if their columns $\hbar^{v_i}, \hbar^{v_j}$ contain the same unordered elements, i.e. $\{\!\{h_{w}^{v_i} : w \in V\}\!\} = \{\!\{h_{w}^{v_j} : w \in V\}\!\}$.

Definition 4.6 (PHI-equivalent nodes)**.** Nodes $v_i, v_j \in V$ are said to be PHI-*equivalent* if and only if they are both row- and column-equivalent.

As we shall prove in Appendix A, the above definitions are not redundant, since row-equivalence and column-equivalence are not comparable, and PHI-equivalence is thus finer than both; moreover, they all refine the equivalence associating nodes with the same rank-list as described in Definition 4.1. Thus, they can serve as criteria to further refine an ordering of the nodes in a graph and, following the above discussion, to further limit the search space for permutation matrices.

Remark 6. PHI-equivalency is a necessary condition for two nodes to belong to the same orbit. If the node partition induced by PHI-equivalency on a graph pair $\{G, H\}$ is discrete, then the same principle of Lemma 3.4 applies and we have a poly-time test for $G \cong H$, since the only possible isomorphism is $\pi(u_i) = w_i$, where $w_i \in V_H$ is the unique PHI-equivalent to $u_i \in V_G$, for all $i \in \{1, \dots, n\}$.

Despite this remark, ordering rows and columns of the matrix (and thus the nodes of the considered graph) may not be trivial: even in the smaller universe of rigid graphs, one can find some interesting examples where the individualisation and the subsequent bisimilarity reduction does not allow any canonical ordering based on the produced hyperset (see Figure 3). On the contrary, when it comes to highly symmetric graphs, the following holds true.

Lemma 4.1. *Let G be a vertex-transitive graph and consider any ordering of its nodes. Then, $\mathsf{PHI}(G)$ is symmetric.*

Proof. Let $G = \langle V, E \rangle$ be a vertex-transitive graph and let $u, w \in V$ be a pair of its nodes. Since there exists an automorphism mapping u onto w, the graphs $\vec{G}_u$ and $\vec{G}_w$ are isomorphic and thus have the same bisimilarity contraction as hypersets, i.e. $h_u = h_w$. Furthermore, since all nodes belong to the same orbit in G, orbits in both $\vec{G}_u$ and $\vec{G}_w$ are solely determined by the distance from the anchors. Therefore, two nodes taken as point and anchor at the same distance will always result in the same pointed membership graph, allowing us to conclude that $h_u^w = h_w^u$. By applying this argument to each pair $u, w \in V$, we conclude that $\mathsf{PHI}(G)$ is symmetric, no matter the ordering of the nodes in V. □

Corollary 4.1. *Let G be a cycle, a clique, the 4×4 Rook's graph or the Shrikhande graph (Figure 4) endowed with whatever ordering of their nodes. Then, $\mathsf{PHI}(G)$ is a symmetric matrix.*

To conclude this section, we prove our claim that PHI-equivalence is a finer relation on the universe of connected simple graphs than the one induced by HI—i.e. by mere hyperset equality—at a local level: PHI-equivalence is then closer to the orbit partition than HI-equivalence.

Lemma 4.2. PHI*-equivalence is strictly more powerful than* HI *at distinguishing non-isomorphic nodes.*

Proof. Consider two nodes u, w. If u and w are PHI-equivalent, it follows that $h_u^u = h_w^w$, as they are the only hypersets whose point and anchor coincide. Thus, $h_u = h_w$ by Remark 5.

To prove that $h_u = h_w$ does not imply that u and w are PHI-equivalent, consider any leaf in both trees T_0^0, T_0^1 of Example 3.2: although their individualisation yields the same hyperset, it is easy to verify that they are not row-equivalent, and thus not PHI-equivalent. □

4.2. Individualisation by the empty set

We define algorithm HIE_k (*Hyperset Individualisation by the Empty set of order k*) on a connected simple graph G. Instead of individualising just a node at a time, we will take k-many, with $k \leq n$, and run maximum bisimulation contraction on the resulting graph, much as HI does. For the sake of completeness, a new node will be linked to the newly-individualised nodes to serve as the point of the resulting hyperset.

Definition 4.7. Let $G = \langle V, E \rangle$ be a connected simple graph, $n = |V|$ and $k \in \{2, \ldots, n\}$, and let $\vec{G}$ be the directed version of G. Then, the *Hyperset Individualisation by the Empty set algorithm of order k* HIE_k is the generalisation of HI obtained by replacing $v \in V$ with $S \subseteq V : |S| = k$ at every occurrence, and $\vec{G}_v$ with

$$\vec{G}_S^E \coloneqq \langle V_G \cup \{v_S\},\ \vec{E}_G \cup \{\langle v_S, v\rangle, \langle v, v_\emptyset\rangle : v \in S\},\ v_S \rangle,$$

at line 7 in Algorithm 1, thus obtaining $\mathsf{HIE}_k(G) \coloneqq \{\!\{h_S^E \coloneqq \mathsf{DPP}(\vec{G}_S^E) : S \subseteq V, |S| = k\}\!\}$.

The addition of v_S, which we will also call *source*, is useful in our set-theoretic framework just to isolate a specific point for the corresponding hyperset h_S^E; thus, $\mathrm{trCl}(h_S^E) \leq n + 1$. Moreover, by applying the same definition to $k = 1$ the source's addition is redundant: by removing it (and dubbing the anchors themselves as the points of the generated hypersets) one gets exactly HI.

Observe that, since each hyperset is associated to a unique k-subset of V, $\mathsf{HIE}_k(G)$ contains $\binom{n}{k}$-many hypersets.

It does not seem easy to see how much the expressive power of HIE_k varies by changing k. However, as a first result, we prove that the expressive power reached by HIE_k on graphs with n-many nodes is the same as the one reached by HIE_{n-k}.

Lemma 4.3 (HIE duality)**.** *Let $G = \langle V_G, E_G \rangle$, $H = \langle V_H, E_H \rangle$ be connected simple graphs. Then, $\mathsf{HIE}_k(G) = \mathsf{HIE}_k(H)$ if and only if $\mathsf{HIE}_{n-k}(G) = \mathsf{HIE}_{n-k}(H)$.*

Proof. Assume $\mathsf{HIE}_k(G) = \mathsf{HIE}_k(H)$: therefore, there is a one-to-one correspondence associating each $S \subseteq V_G$ to a $T \subseteq V_H$, both of cardinality k, in such a way that $h_S^E = h_T^E$. As they are both

contracted by maximum bisimulation, for each node $u \in \mathrm{trCl}(h^E_S) \cup \{h^E_S\}$ there exists exactly one node $w \in \mathrm{trCl}(h^E_T) \cup \{h^E_T\}$ bisimilar to it: notice that u and w are the bisimulation contractions of $\{u' \in V_{\vec{G}^E_S} : u' \equiv_{\vec{G}^E_S} u\}$ and $\{w' \in V_{\vec{H}^E_T} : w' \equiv_{\vec{H}^E_T} w\}$ respectively, and that $u' \in S$ if and only if $w' \in T$. Taking u' and w' as representatives of those bisimilarity classes, and assuming $u' \in S$ and $w' \in T$, by removing the links from the sources v_S, v_T and to the empty set $v_\emptyset$ from both of them in their respective graphs, they will still be bisimilar; analogously, if $u' \in V_G \setminus S$ and $w' \in V_H \setminus T$, by adding the links from the sources and to $v_\emptyset$ to them in their respective graphs they will be bisimilar, as well. Therefore, if u and w are mapped into each other by the isomorphism between h^E_S and h^E_T, then this will be kept when considering u and w within the hypersets $h^E_{V_G \setminus S}$ and $h^E_{V_H \setminus T}$ obtained by individualising the previously non-individualised nodes, thus proving that $h^E_{V_G \setminus S} = h^E_{V_H \setminus T}$ and finally $\mathsf{HIDE}_{n-k}(G) = \mathsf{HIDE}_{n-k}(H)$. The opposite implication is proven in the same way. □

Notice how the previous proof can be summed up by observing that the pointed membership graphs of h^E_S and $h^E_{V \setminus S}$ are the result of computing the maximum bisimulation contraction on the initial partitions $\langle S, V \setminus S\rangle$ and $\langle V \setminus S, S\rangle$ respectively, thus placing the same (bisimilar) nodes together in both hypersets.

Remark 7. Notice that, given $\mathsf{HIE}_k(G)$, we can get $\mathsf{HIE}_{n-k}(G)$ simply by switching the links from the source and to the empty set in every generated hyperset, i.e. it is not necessary to repeat the whole process from scratch as we did above. The resulting hypersets, in general, will not be the same as the previous ones: if $k < n/2$, the number of different distances from the empty set covered by any transitive closure of the resulting hypersets will be higher w.r.t. the one obtained for $n - k$, in which case the hypersets will tend to appear wider but shallower.

As similar considerations apply to the concept of rank—which is the partition DPP-algorithm starts its computation with—it is convenient to individualise the lower possible number of nodes in order to get a finer initial partition of the set of nodes and thus speed up the computation.

The above lemma proves that using a k beyond $\lfloor n/2 \rfloor$ does not increase the expressiveness of HIE_k. Since we have not obtained more precise results about it so far, we can just conjecture that the maximum expressiveness is indeed reached exactly for $k = \lfloor n/2 \rfloor$, although it is possible that a lower k might actually be sufficient. To support our claim, the following example hints that HIE_2 could be strictly more powerful than HI on graphs with sufficiently many nodes.

Example 4.1. The 4×4 Rook's graph and the Shrikhande graph (Figure 4) are distinguished by HIE_2. Indeed, on the Rook's graph, HIE_2 produces two distinct hypersets, obtained by individualising adjacent (48-many copies) and non-adjacent (72-many) node pairs, respectively. On the Shrikhande graph, three distinct hypersets are produced, obtained by individualising adjacent pairs (48-many), non-adjacent pairs with adjacent common neighbours (48-many), and non-adjacent pairs with non-adjacent common neighbours (24-many).

Remark 8. Figure 6 shows the existence of graphs such that there is no single subset $S \subseteq V$ (of any size) whose associated hyperset h^E_S has maximum transitive closure. However, currently, we were not able to find any example of non-isomorphic graph pairs such that the certificates produced by HIE_k are the same for all $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$.

4.3. Individualisation by atoms

In this subsection we introduce yet another alternative version of the basic HI algoritm: HIA_k (*Hyperset Individualisation by Atoms of order k*) on a connected simple graph G. Let $\vec{G}$ be the directed version of G and consider the *gadget graph* proposed in Definition 3.2: in particular, consider its *atoms* $\{v_{\overline{1}}, v_{\overline{2}}, \ldots, v_{\overline{k}}\}$—i.e. its nodes encoding the von Neumann ordinals. By construction, these atoms are pairwise non-bisimilar, each belonging to a unique class: thus, they can be used to enforce non-equivalence on multiple nodes of $\vec{G}$, granting stronger symmetry breaks compared to the previous individualisation methods, which rely upon a single atom $v_\emptyset$.

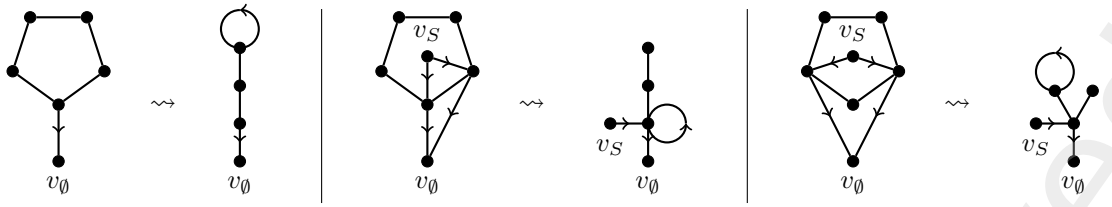


Figure 6: Given a 5-cycle, the depicted situations are the only cases of individualisation and bisimulation contraction that can occur, up to any permutation and taking into account Lemma 4.3. As shown, each case leads to non-trivial bisimulation contractions.

Let $a_i = v_{\bar{i}}$ for each $i \in \{1, \ldots, k\}$. For each set of k distinct nodes $S = \{u_1, \ldots, u_k\} \subseteq V$, define the digraph $\vec{G}^A_S$ by adding arcs $\langle u_i, a_i \rangle$ for $i \in \{1, \ldots, k\}$, and denote by h^A_S the hyperset obtained by collapsing $\vec{G}^A_S$ by maximum bisimulation.

As the only purpose of introducing atoms is to separate nodes which may collapse by bisimulation using HIE_k, the ordering of the nodes in S should be made irrelevant. Given that, we shall introduce an equivalence relation so that different orderings do not affect the reliability of the test.

Definition 4.8. Given k atoms $A = \{a_1, \ldots, a_k\}$ and two hypersets with atoms $h^A_{\{u_1,\ldots,u_k\}}, h^A_{\{w_1,\ldots,w_k\}}$ (in which the i-th node in the subscript is the only one connected by an arc to the atom a_i, for any $i \in \{1, \ldots, k\}$), we write $h^A_{\{u_1,\ldots,u_k\}} \sim_k h^A_{\{w_1,\ldots,w_k\}}$ if and only if $\exists \sigma \in S_k : h^A_{\{u_1,\ldots,u_k\}} = h^A_{\{w_{\sigma(1)},\ldots,w_{\sigma(k)}\}}$, where S_k is the symmetric group over $\{1, \ldots, k\}$.

From this point onwards, with a slight abuse of notation in order to improve readability, we shall use h^A_S to denote the (permutation invariant) equivalence class $[h^A_S]_{\sim_k}$ to which (permutation dependent) hyperset h^A_S belongs.

Definition 4.9. $\mathsf{HIA}_k(G) \coloneqq \{\!\{h^A_S : S \subseteq V \wedge |S| = k\}\!\}$ is the multiset containing (the $\sim_k$ classes of) hypersets h^A_S associated to all subsets of k nodes.

Given this definition, when comparing the multisets produced by HIA_k on a pair of graphs G and H, we will check if the hypersets which they contain are equal *up to any permutation of atoms*. In this way, we ensure that two individualised nodes will be distinguished, but we do not have to consider all the possible permutations of the atoms while performing the algorithm (leading to factorial space complexity), limiting the number of hypersets in $\mathsf{HIDA}_k(G)$ to $\binom{n}{k}$ on a graph G with $n \geq k$ nodes, as HIE_k does. This complexity is eventually transferred to the search space, as, computationally speaking, we do not have an easy way to address sets' comparison without imposing upon them a specific, although arbitrary, ordering. This issue is further discussed in Section 6.

Remark 9. Requiring k additional atoms for HIA_k is actually unnecessary, as we only need $\ell = \lfloor \log k \rfloor + 1$ atoms to produce k different assignment patterns. Indeed, by allowing multiple atoms to be connected to an individualised node, the latter could be identified by the signature binary string of length ℓ having '1' in the i-th position if the node is linked to atom a_i, and '0' otherwise. However, as a trade-off, limiting the number of atoms would force us to increase the number of arcs leading to them in order to produce distinct assignments. Since—in terms of asymptotic complexity—there is no real advantage in doing this, we shall stick to the simpler setting with k distinct atoms, each linked to a single individualised node.

Observe that an analogous of Lemma 4.3 does not hold for HIA_k. Indeed, by individualising k nodes by atoms we obtain an initial partition of the set of nodes in $k + 1$ classes (k of which are singletons), while by individualising $n - k$ of them we always get different initial partitions—they will ultimately lead to the same final partition of the set of nodes originally in G only when it is the discrete partition.

Remark 10. An analogous of Lemma 4.3 would hold only if HIA_k was implemented in a completely different way. If $S_i \subseteq V$ was linked to the atom a_i for $i \in \{1, \ldots, k\}$, $k \leq n$, and $|\bigcup_i S_i| \leq n$, then by replacing each S_i with its complement $V \setminus S_i$ we would get the same final partition by bisimulation.

The following lemma, along with its immediate corollary, allows us to conclude that the HIA family defines a hierarchy of increasingly stronger algorithms, akin to WL_k.

Lemma 4.4. *Given a simple graph $G = \langle V, E\rangle$ such that $|V| = n$ and $k < n$, the multiset $\mathsf{HIA}_{k+1}(G)$ uniquely determines $\mathsf{HIA}_k(G)$, up to any permutation of the atoms.*

Proof. $\mathsf{HIA}_k(G)$ is obtained from $\mathsf{HIA}_{k+1}(G)$ by removing one atom a_i from each hyperset h^A_S and then checking the possible bisimulation between the de-individualised node u_i and any other non-individualised node in h^A_S, thus obtaining $k+1$ new hypersets $h^{A\setminus\{a_i\}}_{S\setminus\{u_i\}}$ for $i \in \{1, \dots, k+1\}$.

As the same subset of k individualised nodes can be obtained from $n-k$ many subsets of cardinality $k+1$, from $|\mathsf{HIA}_{k+1}(G)| = \binom{n}{k+1}$ we get

$$\binom{n}{k+1} \cdot \frac{k+1}{n-k} = \frac{n!}{(k+1)!(n-k-1)!} \cdot \frac{k+1}{n-k} = \binom{n}{k} = |\mathsf{HIA}_k(G)|,$$

thus confirming that the cardinality of the produced multiset coincides with the one of $\mathsf{HIA}_k(G)$. □

Corollary 4.2. *For all $k > 1$, HIA_{k+1} induces a finer or equal partition on the universe of connected simple graphs than HIA_k does.*

As the final point of this preliminary analysis, we prove that the highest expressiveness of HIA_k—which is equivalent to explicitly checking for isomorphism—is reached for $k = n-1$ over graphs with n nodes.

Lemma 4.5. *Let G and H be connected simple graphs with n nodes. Then, the following are equivalent.*

1. $G \cong H$;
2. $\mathsf{HIA}_n(G) = \mathsf{HIA}_n(H)$;
3. $\mathsf{HIA}_{n-1}(G) = \mathsf{HIA}_{n-1}(H)$.

Proof. Trivially, claim 1 implies claims 2 and 3, as the existence of an isomorphism between G and H implies $\mathsf{HIA}_k(G) = \mathsf{HIA}_k(H)$ for every $k \leq n$, in particular for $k = n$ and $k = n-1$.

Assume claim 2 holds true. Then, any bijection $\pi\colon V_G \to V_H$ linking nodes connected to the same atom in two equal hypersets from $\mathsf{HIA}_k(G)$ and $\mathsf{HIA}_k(H)$ is an isomorphism, thus proving claim 1.

Assume claim 3 holds true: we will prove that in this case claim 2 holds, too. Since G and H are connected, each hyperset in both $\mathsf{HIA}_{n-1}(G)$ and $\mathsf{HIA}_{n-1}(H)$ has a unique node that has rank 2 with respect to at least one atom, i.e. it cannot collapse with any other node by computing bisimulation. Therefore, by appending a new atom to this spare node we obtain n copies of the same hyperset (up to any permutation of atoms) from both $\mathsf{HIA}_{n-1}(G)$ and $\mathsf{HIA}_{n-1}(H)$. As these are equal, the resulting hypersets are all $\sim_n$-equivalent, thus proving that $\mathsf{HIA}_n(G) = \mathsf{HIA}_n(H)$. □

4.4. Comparing Expressiveness between HIE and HIA

While the definition of HIE_k comes quite naturally as a generalisation of HI, it is not immediate to see whether adding distinguished atoms (as in HIA_k) provides any advantage. By definition, HIA_k is at least as strong as HIE_k on a local level, since $h^E_S \neq h^E_T \Rightarrow h^A_S \neq h^A_T$ for any two subsets S, T of k nodes, while the opposite implication does not hold true.

Lemma 4.6. *Two pairs of nodes $S = \{u_1, u_2\}, T = \{w_1, w_2\}$ can generate distinct hypersets $h^A_S \neq h^A_T$, but equal hypersets $h^E_S = h^E_T$.*

Proof. Consider Figure 7. Individualisation of u_1 and u_2 in the first graph and of w_1 and w_2 in the second graph yield the same hyperset by doing so with the empty set (HIE_2), but different ones with atoms (HIA_2). The same can be said for pairs $\{u_4, u_5\}$ w.r.t. $\{w_4, w_5\}$ (trivial), and for $\{u_3, u_6\}$ w.r.t. $\{w_3, w_6\}$. □

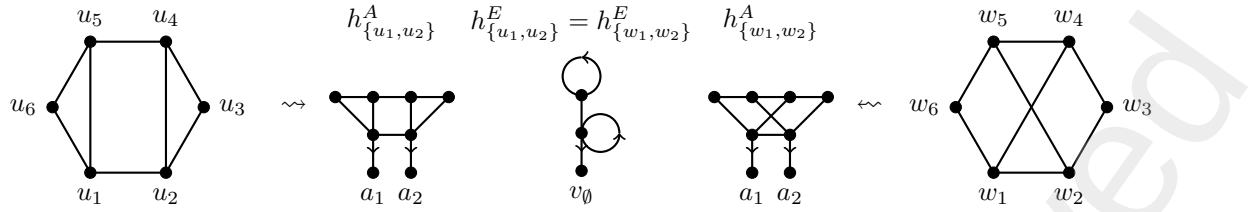


Figure 7: The pair of graph proving Lemma 4.6 (sources have been omitted for simplicity).

It can be shown, by computing the whole $\mathsf{HIE}_k/\mathsf{HIA}_k$ certificates, that the graphs in Figure 7 are distinguished by both methods;[5] thus, although HIA_k is stronger at a local level, it is not necessarily a stronger isomorphism test: HIE_k might still be able to distinguish all graph pairs distinguished by HIA_k, as long as $k \leq \lfloor n/2 \rfloor$. It is yet to be verified whether there exists some k^* such that HIE_{k^*} identifies all n-vertex graphs up to isomorphism: if this holds, the maximum k required for HIA_k would also be at most $k^* \leq \lfloor n/2 \rfloor$, much smaller than the currently established $k \geq n-1$ bound.

5. Hyperset individualisation starting from a given partition

As Example 3.2 and the previous discussion show, HI and its variants (in fact, bisimulation) do not take into account the size of a node's neighbourhood: indeed, it proves itself more powerful than WL_1 exactly when this property is irrelevant, i.e. on regular graphs, but it easily fails otherwise. A natural way to address this limitation, in an attempt to make our algorithms strictly stronger than WL_1, is to endow each node with an label, properly encoded in our set-theoretic framework, in such a way that bisimulation contraction always starts with a given partition of the set of nodes *and* an individualised one. In particular, consider the case in which this initial partition is given by node degrees.

Node degrees can be conveniently represented by adding edges towards a node in a directed, descending chain which is external to the original graph. To do so, we again make use of the gadget graph from Definition 3.2: in particular, we consider its atoms $v_{\{\emptyset\}^k}$ with $k \in \{1, \ldots, n\}$, i.e. the chain encoding the Zermelo ordinals—the *super-singletons* of the empty set. The fact that $v \in V$ has degree d can then be represented by an arc $\langle v, v_{\{\emptyset\}^d} \rangle$.

Definition 5.1. Consider a connected simple graph $G = \langle V_G, E_G \rangle$ and its gadget graph $\vec{N}_G = \langle V_{N_G}, \vec{E}_{N_G} \rangle$ from Definition 3.2. Then, the *Hyperset Individualisation algorithm with Degrees* HID is the variant of HI obtained by re-defining $\vec{G}_v$ as

$$\vec{G}_v = \left\langle V_G \cup V_{N_G},\ \vec{E}_G \cup \vec{E}_{N_G} \cup \{\langle w, v_{\{\emptyset\}^{\deg(w)}} \rangle : w \in V_G\} \cup \{\langle v, v_\emptyset \rangle\},\ v \right\rangle$$

at line 7 in Algorithm 1 (recall that $\vec{E}_G$ is the set of arcs replacing the undirected edges of G).

Notice how this edit shall not create ambiguity with the process of individualising a node: since we are now dealing just with connected graphs, no node is without edges, so they all have a positive degree; therefore, the only arc pointing to the empty set node $v_\emptyset$—now part of the gadget graph—is the one issuing from the intended individualised node.

Remark 11. Let $\vec{G} = \langle V_G \cup V_{\vec{N}_G}, \vec{E}_G \cup \vec{E}_{N_G} \cup \{\langle w, v_{\{\emptyset\}^{\deg(w)}} \rangle : w \in V_G\} \rangle$ (the same as in Definition 5.1, but without individualising any node and with no definite point). Computing maximum bisimulation on $\vec{G}$ is equivalent to computing maximum bisimulation on the graph G by degree-encoding colours.

The following result is an immediate consequence of the new definition.

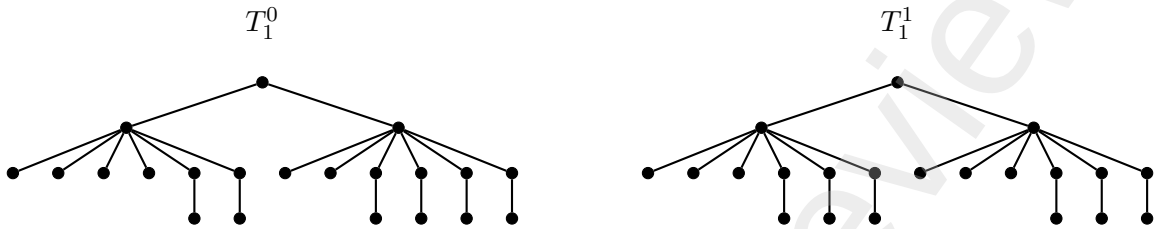
Lemma 5.1. HID *is strictly more expressive than* HI.

[5] By individualising $\{u_1, u_4\}$ or $\{u_1, u_5\}$ w.r.t. $\{w_1, w_4\}$ or $\{w_1, w_5\}$, both HIE_2 and HIA_2 produce different hypersets.

Proof. HI as applied to a connected simple graph G is equivalent to HID where each $\vec{G}_v$ has the degree-encoding arcs pointing the same node in $\vec{N}_G$, apart from $v_\emptyset$. Thus, HID is at least as expressive as HI. Observe that the pair of trees in Example 3.2 are actually distinguished by HID, making it strictly stronger than HI. □

Despite this result, and contrarily to what previously stated in [25, Theorem 4.1], HID fails to be more expressive than WL_1.

Example 5.1. Two trees distinguished by WL_1, but not by HID.



As the previous example shows, endowing our method with a degree-based initial partition is not sufficient for it to capture the counting ability of WL_1 (in this particular case, we are just moving the problem down to the next level of a tree).

Remark 12. Notice how each of the previously presented algorithms (HI, PHI, HIE_k, HIA_k) could be endowed with *any* initial partition in an analogous way, using the gadget defined for HID by taking care to link two nodes of the input graph(s) to the same gadget node if and only if they belong to the same class: this observation, although perhaps trivial to the attentive reader, is crucial for the following theorem and for the next proposed improvement.

Observe that the degree-based partition is the one computed by the first iteration of WL_1. One might wonder whether endowing the graph with the partition computed by WL_1 after a larger, *fixed* number $k \geq 2$ of iterations, before applying hyperset individualisation, could fully retrieve the missing counting ability: the following theorem proves otherwise.

Theorem 5.1. *Let $\mathcal{P}_k(G)$ be the partition induced on a connected simple graph G after the k-th step of WL_1 algorithm. Then, there exists a pair of non-isomorphic trees T_k^0, T_k^1 which cannot be distinguished by hyperset individualisation when taking $\mathcal{P}_k(T_k^i)$ as the initial partition for $i \in \{0, 1\}$.*

Proof. By induction. For $k = 0$ and $k = 1$ consider the instances of tree pairs in Examples 3.2 and 5.1, respectively. Let T_1^0 and T_1^1 be the left and right trees in the latter example.

For greater values of k, define T_k^0 as the tree rooted in node v^0 having two children ℓ^0 and r^0: four copies of T_{k-1}^0 and two copies of T_{k-1}^1 are attached to ℓ^0 by an edge from their roots, while two copies of T_{k-1}^0 and four copies of T_{k-1}^1 are similarly linked to node r^0. T_k^1 is built in an analogous way starting from the root v^1, but the subtrees rooted in ℓ^1 and r^1 are symmetric as they both have three copies of T_{k-1}^0 and three of T_{k-1}^1 as their children. It is easy to see that $T_k^0 \ncong T_k^1$.

By inductive hypothesis, T_{k-1}^0 and T_{k-1}^1 are in the same class w.r.t. $\mathcal{P}_{k-1}(T_{k-1}^0)$ and $\mathcal{P}_{k-1}(T_{k-1}^1)$; in particular, their roots get the same colour. By performing an additional step of the WL_1 algorithm, the roots of such subtrees will get different colours in $\mathcal{P}_k(T_k^0)$ and $\mathcal{P}_k(T_k^1)$. Since T_{k-1}^0 and T_{k-1}^1 both appear the same number of times in T_k^0 and T_k^1, although in different displacements, their colour classes (and their descendants') will have the same cardinality in $\mathcal{P}_k(T_k^0) = \mathcal{P}_k(T_k^1)$. Furthermore, nodes ℓ^0 in T_k^0 and ℓ^1 in T_k^1 will keep the same colours, as an additional step of WL_1, i.e. the $(k+1)$-th one, would be required to distinguish them; the same holds for nodes r^0 and r^1, and consequently also for the tree roots. Observing that nodes of the same colours yield the same hypersets on trees constructed in this way, we can conclude that T_k^0 and T_k^1 are not distinguished by performing individualisation and bisimulation contraction taking $\mathcal{P}_k(T_k^i)$ as the initial partition, thus proving our claim. □

Consequently, if we want HI to be strictly stronger than WL_1, we must endow the graph(s) with the stable colour partition $\mathcal{P}_\infty(\cdot)$ computed by WL_1 at termination: under these conditions, [25, Theorem 4.1] now holds true. We note that computing this finer initial partition would not increase the overall time complexity of our method, as WL_1 takes time $\mathcal{O}(m \log n)$ (see Section 6).

5.1. Iterated hyperset individualisation

In this subsection we put forward a variation on the circle of ideas presented in this paper, aiming at combining bisimulation and atom-introduction for graph canonisation.

Given a graph G, our idea is to *iteratively* and *orderly* equip $\vec{G}$ with atoms, trying to refine bisimulation and "move towards" the discrete partition, thereby associating G to a (possibly unique) hyperset with atoms.

Given G and H to be tested for isomorphism, let $\mathcal{P}(G) = \{C_1, C_2, \ldots, C_k\}$ and $\mathcal{P}(H) = \{C'_1, C'_2, \ldots, C'_k\}$ be two *ordered* partitions of V_G and V_H, respectively, such that for all $\pi \colon V_G \to V_H$:

$$\pi \text{ isomorphism } \Rightarrow \forall i \in \{1, \ldots, k\}, \forall v \in C_i, \pi(v) \in C'_i.$$

In the above hypothesis we say that $\{\mathcal{P}(G), \mathcal{P}(H)\}$ is a *stable pair (of ordered partitions with respect to isomorphism)*. The colouring produced by WL_1 is an example of a stable pair of partitions, since any isomorphism can only map into each other nodes of the same colour (we remark that WL colours can be—and they often are, in fact—ordered at runtime, adopting the approach explained in Section 6).

The following definition introduces a natural way to reflect an initial ordered partition $\mathcal{P}(G)$ by a collection of atoms $\{a_1, \ldots, a_k\}$, added as nodes of some gadget graph with at least n non-bisimilar nodes: again, we assume to rely upon the Zermelo ordinals $v_{\{\emptyset\}^i}$ of gadget graph $\vec{N}_G$, which we previously used to encode degrees in HID.

Definition 5.2. Given a connected simple graph $G = \langle V, E \rangle$ and k atoms $a_1, \ldots, a_k$, the $\{a_1, \ldots, a_k\}$-*rendering* of a partition $\mathcal{P}(G) = \{C_1, \ldots, C_k\}$ is the oriented graph $\vec{G}_{\mathcal{P}(G)} = \langle V \cup V_{N_G}, \vec{E} \cup \vec{E}_{N_G} \cup \{\langle v, a_i \rangle : v \in C_i\} \rangle$.

Remark 13. Although they are constructed in the same way—i.e. by means of some gadget graph—these atoms should not be confused with those at the basis of the previously introduced HIA algorithm, as they perform conceptually different tasks. HIA atoms are used to *enforce* an *arbitrary* distinction between nodes, in order to individualise them and break symmetries; rendering atoms, instead, *depict* an existing partition. These approaches are orthogonal—and they could, in fact, be combined, as we shall explain later in this section: for this reason, we use the Zermelo atoms for partitions and von Neumann atoms for HIA.

The following definition provides some necessary isomorphism conditions to be satisfied by the $\{a_1, \ldots, a_k\}$-renderings of a stable pair $\{\mathcal{P}(G), \mathcal{P}(H)\}$. Specifically, the three conditions require renderings $\{\vec{G}_{\mathcal{P}(G)}, \vec{H}_{\mathcal{P}(H)}\}$ to be bisimilar and their classes to have matching cardinalities and number of connected arcs: if any of the above does not hold, then $G \ncong H$.

Definition 5.3. Given a stable pair $\{\mathcal{P}(G), \mathcal{P}(H)\}$ of $\mathcal{P}(G) = \{C_1, \ldots, C_k\}, \mathcal{P}(H) = \{C'_1, \ldots, C'_k\}$, we say that two $\{a_1, \ldots, a_k\}$-renderings $\vec{G}_{\mathcal{P}(G)}$ and $\vec{H}_{\mathcal{P}(H)}$ are *coherent* if:

1. $\vec{G}_{\mathcal{P}(G)} \equiv \vec{H}_{\mathcal{P}(H)}$;
2. $\forall i \in \{1, \ldots, k\},\ |C_i| = |C'_i|$;
3. $\forall i, j \in \{1, \ldots, k\}, \forall u \in C_i, \forall w \in C'_i,\ |\{\langle u, v \rangle : v \in C_j\}| = |\{\langle w, v \rangle : v \in C'_j\}|$.

Remark 14. Coherence of two renderings can be verified in polynomial time. Condition 2 and 3 of Definition 5.3 are trivially poly-time, while condition 1 can be checked by a poly-time bisimulation test (see Section 6).

Notice how conditions 2 and 3 guarantee the partitions to be *at least* as fine as the stable colourings computed by WL_1.

We propose an algorithm that iteratively refines an initial pair of coherent renderings $\{\vec{G}_{\mathcal{P}(G)}, \vec{H}_{\mathcal{P}(H)}\}$ into another pair of renderings $\{\vec{G}_{\hat{\mathcal{P}}(G)}, \vec{H}_{\hat{\mathcal{P}}(H)}\}$ resulting in one of three possible outcomes:

1. $\{\vec{G}_{\hat{\mathcal{P}}(G)}, \vec{H}_{\hat{\mathcal{P}}(H)}\}$ are not coherent, and thus $G \ncong H$;
2. $\{\vec{G}_{\hat{\mathcal{P}}(G)}, \vec{H}_{\hat{\mathcal{P}}(H)}\}$ are coherent and discrete. Since they are discrete—thus, $\{\hat{\mathcal{P}}(G), \hat{\mathcal{P}}(H)\}$ are the orbits of rigid graphs—and by definition of coherent renderings, it must be $G \cong H$;
3. $\{\vec{G}_{\hat{\mathcal{P}}(G)}, \vec{H}_{\hat{\mathcal{P}}(H)}\}$ are coherent, not discrete and cannot be refined any further. In this case, candidate isomorphisms must be checked only among those mappings that respect the resulting stable pair $\{\hat{\mathcal{P}}(G), \hat{\mathcal{P}}(H)\}$.

In the *Iterated Hyperset Individualization algorithm* below, we assume to launch Algorithm 1 with a generic initial partition.

Algorithm 2 Iterated Hypersets Individualization

Require: Coherent $\{a_1, \ldots, a_k\}$-renderings $\vec{G}_{\mathcal{P}(G)}$ and $\vec{H}_{\mathcal{P}(H)}$.
Ensure: Coherent $\{a_1, \ldots, a_{\hat{k}}\}$-renderings $\vec{G}_{\hat{\mathcal{P}}(G)}$ and $\vec{H}_{\hat{\mathcal{P}}(H)}$, with $\hat{k} \geq k$, or $G \ncong H$

1: $\hat{k} \leftarrow 0$
2: $\hat{\mathcal{P}}(G) \leftarrow \mathcal{P}(G), \hat{\mathcal{P}}(H) \leftarrow \mathcal{P}(H)$
3: **while** $\hat{k} \neq k$ **do** ▷ iterate while proper refinements are produced
4: $\quad k \leftarrow |\hat{\mathcal{P}}(G)|$ ▷ store the current number of classes
5: $\quad \mathcal{P}(G) \leftarrow \hat{\mathcal{P}}(G), \mathcal{P}(H) \leftarrow \hat{\mathcal{P}}(H)$ ▷ store the current partitions
6: $\quad$ **for** $T \in \{G, H\}$ **do**
7: $\quad\quad$ let $\hat{\mathcal{P}}(T)$ be the *ordered* partition induced by $\mathsf{HI}(T)$ with initial partition $\mathcal{P}(T)$
8: $\quad\quad$ let $\vec{T}_{\hat{\mathcal{P}}(T)}$ be the $\{a_1, \ldots, a_{\hat{k}(T)}\}$-rendering of $\hat{\mathcal{P}}(T)$ ▷ set the new rendering
9: $\quad$ **end for**
10: $\quad$ **if** $\vec{G}_{\hat{\mathcal{P}}(G)}$ and $\vec{H}_{\hat{\mathcal{P}}(H)}$ are not coherent **then** ▷ check for coherence
11: $\quad\quad G \ncong H$ ▷ declare non isomorphic if different hypersets are produced
12: $\quad$ **end if**
13: $\quad \hat{k} \leftarrow \hat{k}(G)$ ▷ store new number of classes—note: $\hat{k}(G) = \hat{k}(H)$
14: **end while**

A delicate step in Algorithm 2 is instruction 7: for $T \in \{G, H\}$, the computation of the ordered version of the refined partition induced by HI when launched with initial partition $\mathcal{P}(T)$. To obtain an univocal ordering of the refined partition we can, for example, launch HI on both graphs in parallel and perform class refinements in exactly the same order on both G and H. This idea is the one already mentioned in Remark 14 and illustrated in Section 6 below for testing apg-bisimilarity (Algorithm 3).

An alternative way for producing ordered partitions is to introduce a universal uniform code for hypersets, such as the one proposed in [24].

Note that HI, called at instruction 7 of Algorithm 2, could be replaced by any method for computing a finer partition on the graphs. In particular, one could use the previously introduced higher-dimensional hyperset individualisation algorithms, HIE_k and HIA_k. In the case of HIA_k, one must only take care to use a separate set of atoms—the von Neumann ordinals $v_{\overline{k}}$ of $\vec{N}_G$—to separate the k individualised nodes; furthermore, since $\{\emptyset\}^1 = \overline{1} = \{\emptyset\}$, an additional node should be added to either the Zermelo or the von Neumann ordinals section of the gadget graph, in order to avoid overlap.

Remark 15. An equivalent version of Lemma 3.3 also holds in this case: whenever a hyperset h of maximum transitive closure is found for one of the graphs, then $G \cong H$ if and only if h also belongs to the certificate of the other graph. This check can be easily integrated into the algorithm.

To conclude this section, we revisit the third possible outcome of Algorithm 2, where the renderings of the two final partitions $\hat{\mathcal{P}}(G) = \{C_1, \ldots, C_{\hat{k}}\}$ and $\hat{\mathcal{P}}(H) = \{C'_1, \ldots, C'_{\hat{k}}\}$ are coherent, but not

discrete. A necessary condition for G and H to be isomorphic is that the subgraphs induced by the i-th classes of the final partitions, say $G|_{C_i}$ and $H|_{C'_i}$, must themselves be isomorphic for all $i \in \{1, \dots, \hat{k}\}$. This observation naturally suggests a recursive, *divide et impera* approach to the problem: after applying the iterative algorithm to the pair $\{G, H\}$, one can recursively process pairs of non-singleton subgraphs $\{G|_{C_i}, H|_{C'_i}\}$, possibly leveraging a stronger version of the underlying hyperset individualization subroutine (e.g. HIA_k in place of HI) to further refine the partition. This approach is going to be explored in future works.

6. Implementations and Complexity

We now provide an analysis of the time and space complexity of the aforementioned methods; as a reference, we recall that WL_k has time complexity $\mathcal{O}(n^{k+1} \log n)$. Although a multiset of hypersets may seem harder to describe than the multiset of colours produced by WL_k, it must be noted that such colours are iteratively obtained by applying a hash function to a multiset of previously computed colours, i.e. for all purposes they are equivalent to nested multisets.

Multisets of hypersets can be handled, for instance, by keeping a list L of all distinct hypersets generated during a run of HI on a graph $G = \langle V, E \rangle$. Whenever a node $v \in V$ is individualised, $h_v \in L$ can be checked in polynomial time [14]: if this is the case, we simply increase by 1 the multiplicity of h_v in $\mathsf{HI}(G)$; otherwise, we add h_v to L and $\mathsf{HI}(G)$, with multiplicity 1.

A single hyperset h_v can be computed in time $\mathcal{O}(m \log n)$ by an algorithm such as [8]. Whether h_v belongs to L can be checked in time $\mathcal{O}(|L|\ m \log n)$, where $|L| \leq n$ is the number of unique hypersets in L, again using [8]. Algorithm 3 shows how to compare hypersets by their pointed membership graphs, running bisimulation on a third apg: such procedure can then be applied to h_v and each $h \in L$.

Algorithm 3 Hypersets comparison

Require: $\vec{G}_1 = \langle V_1, \vec{E}_1, p_1 \rangle, \vec{G}_2 = \langle V_2, \vec{E}_2, p_2 \rangle$ ▷ Pointed membership graphs h_1, h_2 to be compared
Ensure: $h_1 \stackrel{?}{=} h_2$
1: $V_0 \leftarrow V_1 \cup V_2 \cup \{p_0\}$
2: $\vec{E}_0 \leftarrow \vec{E}_1 \cup \vec{E}_2 \cup \{\langle p_0, p_1 \rangle, \langle p_0, p_2 \rangle\}$
3: $\vec{G}_0 \leftarrow \langle V_0, \vec{E}_0, p_0 \rangle$ ▷ New apg whose point is linked to the points of $\vec{G}_1$ and $\vec{G}_2$
4: $h_0 \leftarrow \mathsf{DPP}(\vec{G}_0)$
5: **if** $p_1 \equiv_{\vec{G}_0} p_2$ **then** ▷ Equivalently, they are merged together in h_0
6: $\quad h_1 = h_2$ ▷ h_1 and h_2 are bisimilar, and thus equal
7: **else**
8: $\quad h_1 \neq h_2$
9: **end if**

Each $h \in L$ may also be associated to a unique integer in $\{1, \dots, n\}$, according to the time it first appeared in L, allowing for constant time comparisons between the hypersets of distinct nodes, if later needed.

To test a graph-pair $\{G, H\}$ for isomorphism, one can simply run HI in parallel on both, taking care of checking both L_G and L_H whenever a hyperset h_v from either graph is computed, in order to keep the aforementioned hyperset enumeration coherent. In terms of space, this solution requires $\mathcal{O}(n + m)$ for each $h \in L_G \cup L_H$, by definition of bisimilarity: in the worst case, assuming each individualised hyperset to be distinct, HI has space complexity $\mathcal{O}(n(n + m))$. In terms of time, the overall complexity of HI is $\mathcal{O}(n^2 m \log n)$, since up to n^2 hypersets comparisons, each of cost $\mathcal{O}(m \log n)$, may be required if either $|L_G| \in \Theta(n)$ or $|L_H| \in \Theta(n)$.

In the case of PHI, time complexity remains the same, since this variant computes the exact same n hypersets as HI—one for each anchor, i.e. for each row of the matrix. One must only take care to store the class of each node (point) in the bisimilarity reductions of each anchored graph: this can be achieved by minor modifications of the DPP algorithm [8].

Moving to the k-dimensional HIE_k, complexity scales according to the number of computed hypersets: in the worst cases, space $\mathcal{O}(\binom{n}{k}(n+m))$ and time $\mathcal{O}(\binom{n}{k}^2 m \log n)$ may be required. HIA_k may appear equally costly, since its final multiset also contains $\binom{n}{k}$ elements; however, it must be noted that this relies upon the use of equivalence relation $\sim_k$, which hides the computational cost of comparing two hypersets $h^A_{\{u_1,\dots,u_k\}}$ and $h^A_{\{w_1,\dots,w_k\}}$ up to any of the $k!$ possible permutations of the atoms. Since checking whether two hypersets belong to the same $\sim_k$ class requires $k!$ comparisons of the kind described in Algorithm 3, the overall time complexity of HIA_k is $\mathcal{O}(k!\binom{n}{k}^2 m \log n)$.

Encoding any initial partition requires adding a linear number of nodes and arcs, which does not alter the complexity of any of the above methods.

The iterative method proposed in Section 5 adds an n multiplicative factor to the complexity of each proposed method, as at most n iterations of the refinement process can occur before the algorithm comes to a stop.

To conclude this section, we point out how, on a practical level, simple heuristics can be applied to avoid costly comparisons in all of the above algorithms: for instance, hypersets whose pointed membership graphs differ in their number of nodes or edges will certainly be distinct.

6.1. HIE_k on CFI graphs

In this section, we consider some tests run with HIE_k on CFI graphs [4], performed by using the tool provided by Alessandro Minisini and available at the URL `https://github.com/alemini18/hid_algos`.[6]

Example 6.1. Consider Example 3.1, recalling that the provided graphs G and H are distinguished by HI but not by WL_1. We will:

1. expand the graphs by CFI construction, thus obtaining $X(G), \tilde{X}(G), X(H), \tilde{X}(H)$ (for simplicity, the twisted edge will be the middle one—however, as reminded in Section 2.2, one can choose whatever edge);
2. reduce the obtained graphs by keeping the only interesting features (i.e. reducing the external paths to single edges): we will name the resulting graphs $X^r(G), \tilde{X}^r(G), X^r(H), \tilde{X}^r(H)$;
3. run HIE_k on them for varying k.

The resulting graphs are pictured in Figure 8: they are all 3-regular with the same number of nodes, so the four of them are not distinguished by colour refinement. The results for what concerns HIE_k are that:

- for $k = 1$, HI is able to tell apart $\tilde{X}^r(H)$ from $X^r(H)$, but *not* $\tilde{X}^r(G)$ from $X^r(G)$;
- for $k = 2$ (and possibly higher) the four of them are kept distinguished from one another.

Observe that G has a separator of cardinality 1 (remove one of the middle vertices), while for H the minimum cardinality of a separator is 2. It can be checked by hand.

This first result might already suggest that the HIE_k and WL_k families are not actually related. To bring further evidence to this claim, we try to push the above example to a higher level.

Indeed, consider a family of (CFI-like) graphs such that, for each $d \in \mathbb{N} \setminus \{0, 1\}$, is constituted by two copies of X_d, plus each link between a port of the first one and a port of the second one; the case $d = 3$ is $X(H)$ of Example 6.1. Observe that each graph G_d suitably represents the CFI expansion of a graph with a minimum separator of cardinality d.

By running our tests for $d < 10$, we observed that each so-built G_d is distinguished from its twisted version $\tilde{G}_d$ even by HI. An easy explanation is that, with such a construction, twisting a pair of edges

[6] To be precise, the implemented method is HIDE_k (see [25]). However, as the considered graphs are all regular, the preprocessing by node degrees is futile.

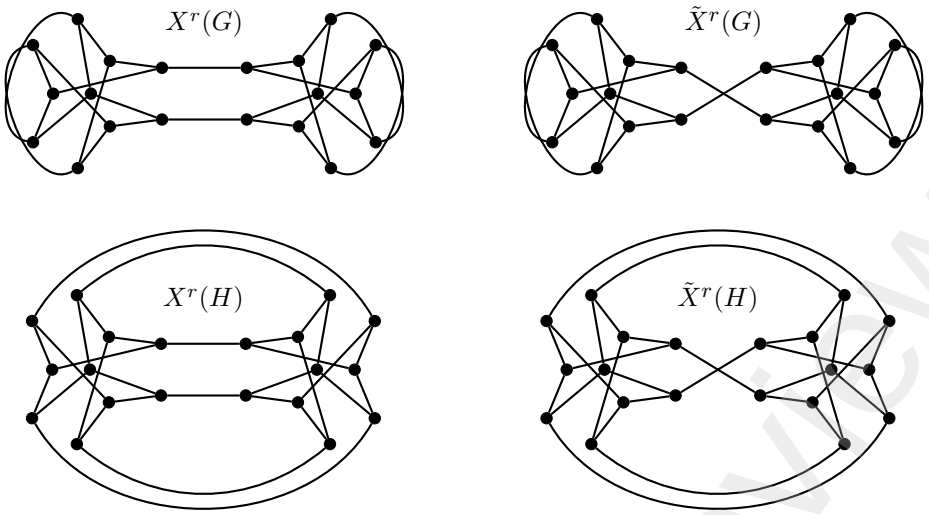


Figure 8: Graphs from Example 6.1.

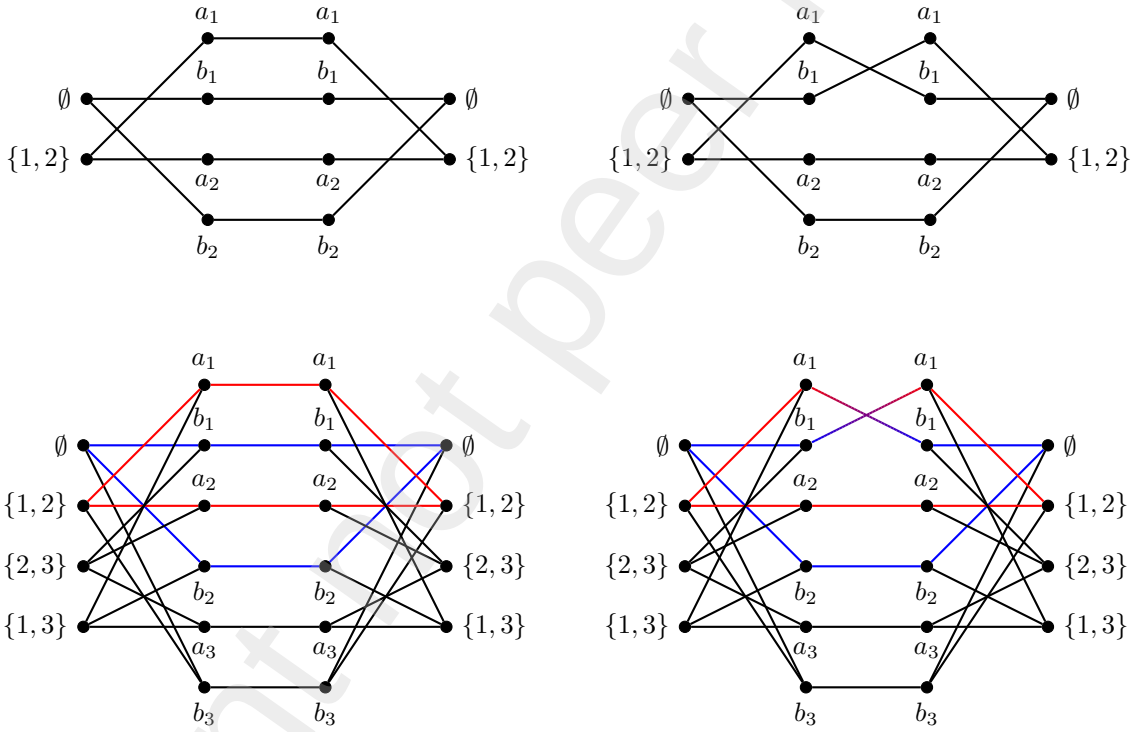


Figure 9: Top: two copies of X_2 connected by two pairs of straight edges (left) and by twisting one of these pairs (right). Observe that the two separate cycles of length 6 are merged in a single cycle with length 12, resulting in different HI-certificates for the two graphs. Bottom: analogous considerations can be applied to this graph-pair (isomorphic to $X^r(H), \tilde{X}^{r^{(H)}}$ of Example 6.1—this view allows a clearer identification of cycles).

in the described way affects the length of cycles crossing two ports; this can be visualised in Figure 9. As HI is rank-sensitive from every possible "point of view" (individualised node), the non-isomorphic pair is immediately spotted by our method.

7. Open Problems and Conclusions

In this paper we illustrated a line of attack to the Graph Isomorphism problem that makes use of the notion of hyperset, combined with the matching notion of bisimulation as equality criterion among hypersets. In doing so we compared our proposal with *colour refinement*—the celebrated 1-dimensional Weisfeiler-Leman algorithm, WL_1—and presented our approach as an integration of multi-point individualisation with bisimulation reduction. The resulting family of algorithms we were able to put forward produces multisets of hypersets for a given graph—acting as one-way error isomorphism certificates—and paves the way to a rather detailed (and hopefully promising) characterisation of the equivalence relations resulting from hyperset computation. Classes of these equivalence relations are univocally associated to the elements (themselves hypersets) in the transitive closures of the hypersets obtained when different nodes are used as *anchors*. This novel spin on the GI problem allows us to look at colour refinement from a set-theoretic perspective. In fact, the node partition induced by our bisimulation techniques can be "rendered" in the form of atoms (*urelements*) introduced as fresh objects/elements into hypersets: this allows the bisimulation-based refinement procedure to be restarted, in an iterative fashion, until a fix-point stable partition is reached.

A number of questions remain open. For example, how many nodes do we need to *anchor* concurrently, in general, in order to guarantee that any pair of non-isomorphic graphs with n nodes will be distinguished by HIE_k (assuming that such a feat is achievable)? We observed in Remark 8 that we have not been able to find a pair of non-isomorphic graphs $\{G, H\}$ such that $\{\mathsf{HIE}_k(G) : k \in \{0, \ldots, \lfloor n/2 \rfloor\}\} = \{\mathsf{HIE}_k(H) : k \in \{0, \ldots, \lfloor n/2 \rfloor\}\}$.

How does the picture change if we allow multiple atoms (i.e. we use HIA_k instead of HIE_k) in place of using just the empty set $\emptyset$? And, moreover, is the design of a divide-and-conquer approach integrable within the circle of ideas presented in 5.1, where we discussed hyperset individualisation by multiple atoms in an iterated format?

Concerning the architecture of algorithms based on the conceptual framework presented here, two further questions arise. First, we ask whether the usage of *graded* bisimulation is necessary in our setting—in order for an algorithm to attain *counting* ability—or whether it could be replaced by pure bisimulation (and, in the latter case, at what cost in terms of complexity). Second, considering the iterative technique illustrated above, we wonder how the approach could be further expanded by including *recursive* calls to a more expressive GI solver—for instance, a higher dimensional HIA_k—on the subgraphs induced by (pairs of) equivalence classes computed at the end of Algorithm 2, in order to either establish $G \ncong H$ or to get a finer initial rendering from which the iterative process could be restarted.

Aknwoledgements

The authors would like to thank Alessandro Minisini (Università degli Studi di Udine) for implementing the method HIDE_k, thus aiding to test some sample graphs and to develop Section 6.1.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] B. Weisfeiler, A. A. Lehman, A Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction, Nauchno-Technicheskaya Informatsia Ser. 2 (1968) 12–16.

[2] L. Babai, R. Mathon, Talk at the south-east conference on combinatorics and graph theory, 1980.

[3] L. Babai, P. Erdös, S. Selkow, Random graph isomorphism, SIAM J. Comput. 9 (1980) 628–635. doi:10.1137/0209047.

[4] J.-Y. Cai, M. Furer, N. Immerman, An optimal lower bound on the number of variables for graph identification, in: 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 612–617. doi:10.1109/SFCS.1989.63543.

[5] M. Grohe, D. Neuen, Recent advances on the graph isomorphism problem, London Mathematical Society Lecture Note Series, Cambridge University Press, 2021, p. 187–234.

[6] F. Fuhlbrück, J. Köbler, I. Ponomarenko, O. Verbitsky, The Weisfeiler-Leman algorithm and recognition of graph properties, in: T. Calamoneri, F. Corò (Eds.), Algorithms and Complexity, Springer International Publishing, Cham, 2021, pp. 245–257.

[7] B. D. McKay, A. Piperno, Practical graph isomorphism, ii, Journal of Symbolic Computation 60 (2014) 94–112. URL: https://www.sciencedirect.com/science/article/pii/S0747717113001193. doi:https://doi.org/10.1016/j.jsc.2013.09.003.

[8] A. Dovier, C. Piazza, A. Policriti, An efficient algorithm for computing bisimulation equivalence, Theoretical Computer Science 311 (2004) 221–256. URL: https://www.sciencedirect.com/science/article/pii/S030439750300361X. doi:https://doi.org/10.1016/S0304-3975(03)00361-X.

[9] E. G. Omodeo, Bisimilarity, hypersets, and stable partitioning: a survey, Rend. Istit. Mat. Univ. Trieste Volume 42 (2010) 211–234.

[10] M. Ajtai, Recursive construction for 3-regular expanders, in: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), 1987, pp. 295–304. doi:10.1109/SFCS.1987.50.

[11] T. Jech, Set Theory: The Third Millennium Edition, revised and expanded, Springer Monographs in Mathematics, 3 ed., Springer Berlin Heidelberg, 2003. URL: https://books.google.it/books?id=CZb-CAAAQBAJ.

[12] M. Forti, F. Honsell, Set theory with free construction principles, Annali della Scuola Normale Superiore di Pisa - Classe di Scienze 10 (1983) 493–522. URL: http://eudml.org/doc/83914.

[13] P. Aczel, Non-Well-Founded Sets, Csli Lecture Notes, Palo Alto, CA, USA, 1988.

[14] E. G. Omodeo, A. Policriti, A. I. Tomescu, On Sets and Graphs: Perspectives on Logic and Combinatorics, Springer, 2017. URL: https://link.springer.com/book/10.1007/978-3-319-54981-1. doi:10.1007/978-3-319-54981-1.

[15] M. de Rijke, A note on graded modal logic, Studia Logica: An International Journal for Symbolic Logic 64 (2000) 271–283. URL: http://www.jstor.org/stable/20016145.

[16] I. N. Ponomarenko, The isomorphism problem for classes of graphs closed under contraction, Journal of Soviet Mathematics 55 (1991).

[17] M. Grohe, D. Neuen, D. Wiebking, Isomorphism testing for graphs excluding small minors, SIAM Journal on Computing 52 (2023) 238–272. URL: https://doi.org/10.1137/21M1401930. doi:10.1137/21M1401930. arXiv:https://doi.org/10.1137/21M1401930.

[18] R. Frucht, Graphs of degree three with a given abstract group, Canadian Journal of Mathematics 1 (1949) 365–378. doi:10.4153/CJM-1949-033-6.

[19] G. Baron, W. Imrich, Asymmetrische reguläre graphen, Acta Mathematica Academiae Scientiarum Hungarica 20 (1969) 135–142. URL: https://doi.org/10.1007/BF01894574. doi:10.1007/BF01894574.

[20] V. Arvind, F. Fuhlbrück, J. Köbler, O. Verbitsky, On Weisfeiler-Leman invariance: Subgraph counts and related graph properties, Journal of Computer and System Sciences 113 (2020) 42–59. URL: https://www.sciencedirect.com/science/article/pii/S0022000020300386. doi:https://doi.org/10.1016/j.jcss.2020.04.003.

[21] M. Fürer, On the combinatorial power of the Weisfeiler-Lehman algorithm, in: D. Fotakis, A. Pagourtzis, V. T. Paschos (Eds.), Algorithms and Complexity, Springer International Publishing, Cham, 2017, pp. 260–271.

[22] A. Brouwer, W. Haemers, Spectra of Graphs, Springer New York, 2012. doi:10.1007/978-1-4614-1939-6.

[23] G. D'Agostino, E. G. Omodeo, A. Policriti, A. Tomescu, Mapping sets and hypersets into numbers, Fundamenta Informaticae 140 (2015) 307–328. doi:10.3233/FI-2015-1256.

[24] S. Boscaratto, D. Cantone, E. G. Omodeo, A. Policriti, The ackermann encoding and its siblings, Journal of Logic and Computation 36 (2026) exaf070. URL: https://doi.org/10.1093/logcom/exaf070.

doi:10.1093/logcom/exaf070.
[25] S. Boscaratto, F. Nascimben, A. Policriti, Hyperset individualisation algorithms, in: L. Moscardelli, F. Scozzari (Eds.), Proceedings of the 26th Italian Conference on Theoretical Computer Science, Pescara, Italy, September 10-12, 2025, volume 4039 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 67–80. URL: https://ceur-ws.org/Vol-4039/paper25.pdf.

A. Equivalency comparison

In this appendix, we shall compare different equivalence relations introduced along the paper, in particular the ones defined in Section 4.1.

We first define the equivalence induced by the non-totality of the pre-order of Definition 4.1.

Definition A.1 (Rank-equivalence)**.** Two nodes u, w in a graph are said to be *rank-equivalent* if their rank-lists ℓ_u, ℓ_w are the same.

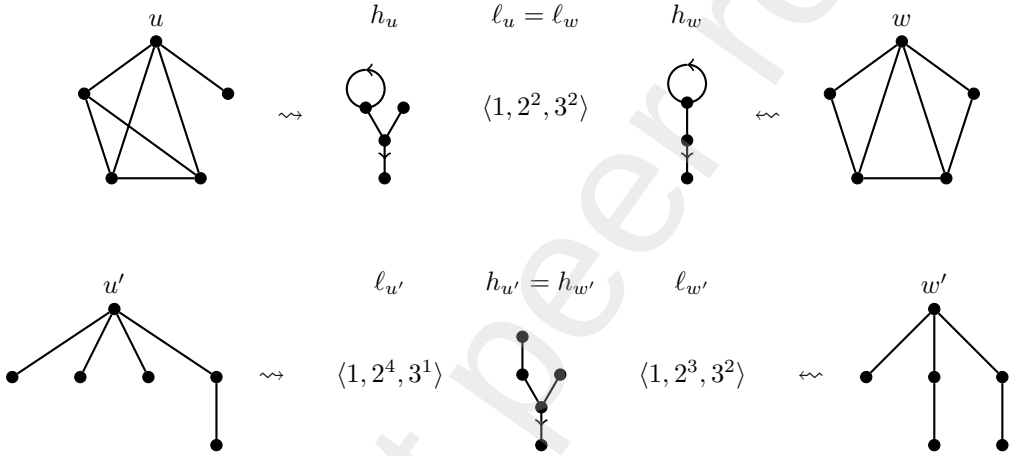


Figure 10: Two graph-pairs showing that HI- and rank-equivalence are incomparable. Top: nodes u, w are rank-equivalent, but not HI-equivalent; bottom: nodes u', w' are HI-equivalent, but not rank-equivalent.

Lemma A.1. HI*-equivalence and rank-equivalence are incomparable.*

Proof. Consider Figure 10: nodes u and w are rank-equivalent, but not HI-equivalent; on the contrary, u' and w' are HI-equivalent without being rank-equivalent (see picture). □

Recall that two PHI-equivalent nodes are HI-equivalent (Lemma 4.2); in particular, both row- and column-equivalency imply HI-equivalency.

Lemma A.2. *Row- and column-equivalence are strictly finer than* HI*-equivalence.*

Proof. For what concerns row-equivalence, see the proof of Lemma 4.2. The same argument can be used to prove that two leaves from the distinct trees are also non-column-equivalent. □

Lemma A.3. *Row- and column-equivalence are strictly finer than rank-equivalence.*

Proof. Two row-equivalent nodes u, w must have the same number of nodes in each bisimilarity class of $h_u = h_w$; this implies, in particular, that the number of nodes at each rank must coincide (thus $\ell_u = \ell_w$). A very similar argument can be used to prove that column-equivalence is also at least as fine as rank-equivalence.

Consider again nodes u and w in Figure 10: although rank-equivalent, they are neither row- nor column-equivalent, thus ending the proof. □

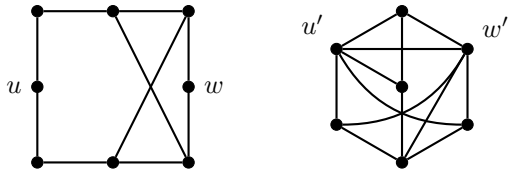


Figure 11: Graph pair showing that row- and column-equivalence are incomparable. Left: nodes u, w are row-equivalent, but not column-equivalent; right: nodes u', w' are column-equivalent, but not row-equivalent.

The next result is the most meaningful, as it proves that PHI-equivalence introduced in Definition 4.6 does not coincide with any of the previous relations.

Lemma A.4. *Row-equivalence and column-equivalence are incomparable.*

Proof. Consider Figure 11: nodes u and w are row-equivalent, but not column-equivalent; on the contrary, nodes u' and w' are column-equivalent without being row-equivalent. □

Corollary A.1. PHI-*equivalence is a finer relation than row- or column-equivalence alone.*