A Systematic Approach to Universal Random Features in Graph Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Universal random features (URF) are state of the art regarding practical graph neural networks that are provably universal. There is great diversity regarding terminology, methodology, benchmarks, and evaluation metrics used among existing URF. Not only does this make it increasingly difficult for practitioners to decide which technique to apply to a given problem, but it also stands in the way of systematic improvements. We propose a new comprehensive framework that captures all previous URF techniques. On the theoretical side, among other results, we formally prove that under natural conditions all instantiations of our framework are universal. The framework thus provides a new simple technique to prove universality results. On the practical side, we develop a method to systematically and automatically train URF. This in turn enables us to impartially and objectively compare all existing URF. New URF naturally emerge from our approach, and our experiments demonstrate that they improve the state of the art.

1 Introduction

Structured data is omnipresent in the modern world. Graphs are a fundamental data type in machine learning (ML) on structured data and used in a variety of learning algorithms (Shervashidze et al., 2011; Nickel et al., 2016; Hamilton et al., 2017), including graph neural networks (GNNs) (Zhou et al., 2020; Wu et al., 2021). GNNs, and especially message passing neural networks (MPNNs) (Gilmer et al., 2017), have found widespread applications, from drug design to friendship recommendation in social networks (Song et al., 2019; Tang et al., 2020). However, severe limitations of MPNNs have recently been proven formally. In fact, MPNNs are at most as expressive as *color refinement* (Xu et al., 2019; Morris et al., 2019). Color refinement — also known as the 1-dimensional Weisfeiler-Leman algorithm — is a simple algorithm inducing a *non*-universal similarity measure on graphs. Thus MPNNs fail to be universal function approximators, a fundamental property known for multilayer perceptrons since the 1980s (Cybenko, 1989).

Addressing this lack, universal random features (URF) (Murphy et al., 2019; Dasoulas et al., 2020; Sato et al., 2021; Abboud et al., 2021) were developed, which provably enable MPNNs to be universal function approximators. URF enhances the nodes of the graph with random values, thus artificially breaking its intrinsic symmetries and facilitating the distinction of previously indistinguishable nodes. URF represent the state of the art in practical, efficient, and universal GNNs. Recently, numerous variations of URF have been developed (Murphy et al., 2019; Dasoulas et al., 2020; Sato et al., 2021). No systematic comparison of these URF is available and it is unclear how their trainability, generalizability, and expressivity (Raghu et al., 2017) compare to one another, because of inconsistent terminology, differing or incomplete data sets for benchmarking, and orthogonal evaluation metrics. This makes it difficult for practitioners to decide which technique to apply. It also stands in the way of systematic improvements.

Our contributions. We propose two systematic approaches to dealing with URF: 1) The individualization-refinement node initialization (IRNI) framework, a theoretical description that encompasses all currently known URF schemes (and even proposes new ones, see Figure 1). The foundation for our framework is graph isomorphism theory, particularly the individualization-refinement (IR) paradigm. 2) A tuning technique

based on bayesian hyperparamter optimization (BHO), designed specifically to tune methods based on the IRNI framework.

Armed with these new systematic approaches, we make the following contributions:

- We demonstrate that our framework IRNI is comprehensive. In fact, we show that all previous URF techniques are captured.
- We formally prove that all instantiations of the IRNI framework that satisfy a natural compatibility condition are universal and equivariant, even ones not considered so far. Furthermore, we prove that already a very limited version of IRNI is universal on almost all graphs, including all 3-connected planar ones. We also quantify the amount of randomness required to ensure universality of all but an exponentially small fraction of all graphs.
- We apply and systematically test "ensembling over randomness" on all URF. This is a particular form of ensembling that previously has been applied in some but not all URF (Murphy et al., 2019; Dasoulas et al., 2020). Our crucial new insight is that ensembling over randomness significantly improves the performance of all URF, even in cases in which it had not been considered previously.
- The IRNI framework also suggests a new most natural URF directly related to practical graph isomorphism solvers.
- We compare all previous URF methods using the same BHO tuning approach. This leads to multiple new state-of-the-art performances. In particular, all previous URF methods are improved upon. Additionally, we discover that even though random node initialization (RNI) was reported to be harder to train than other URF (Abboud et al., 2021), our tuning approach resolves this issue.

In our evaluation, and somewhat surprisingly, we find that currently, there seems to be no clear best strategy in practice. There are strengths and weaknesses to all considered URF. However, this marks the first direct comparison of all the different URF and thus enables a more informed choice of which method to use in practice.

Our hope is that our systematic approach also eases future development of URF, in particular for developing and evaluating new algorithmic ideas.

Individualization refinement in a nutshell: The individualization-refinement framework is a general technique to devise algorithms for tasks revolving around symmetry. Specifically, it can be used to compute isomorphisms, automorphisms (or symmetries), and canonical forms of graphs or other combinatorial objects. The idea is first to use efficient subroutines to distinguish vertices according to simple structural properties preserved by symmetries. An example, which in fact, is similar to what is computed by color refinement (also known as the 1-dimensional Weisfeiler-Leman algorithm), iteratively collects information on the degree of neighbors of a vertex and the degrees of the neighbors of the neighbors and so on. These efficient subroutines must be isomorphism invariant and refine the partition of vertices into indistinguishable parts (and are therefore called refinements).

Once a state is reached where refinements no longer yield new information, an IR algorithm artificially distinguishes a vertex v with a so-called individualization. This individualization breaks apparent symmetry and allows us to recurse by applying another refinement step, and so on. Since the individualization is artificial, to preserve isomorphism-invariance we also have to perform the individualization in a backtracking manner with all vertices which have not been distinguished from the vertex v at this point. The backtracking process finishes once all vertices have been distinguished, making symmetry computations trivial. We refer to section 4 for a more in-depth explanation of the IR machinery necessary for this work.

Why individualization refinement. We explain why the IR-framework is a natural choice for the development of efficient universal GNNs. (For general strengths and weaknesses beyond ML see (McKay & Piperno, 2014; Neuen & Schweitzer, 2017).) First of all, the introduction of graph isomorphism techniques in the

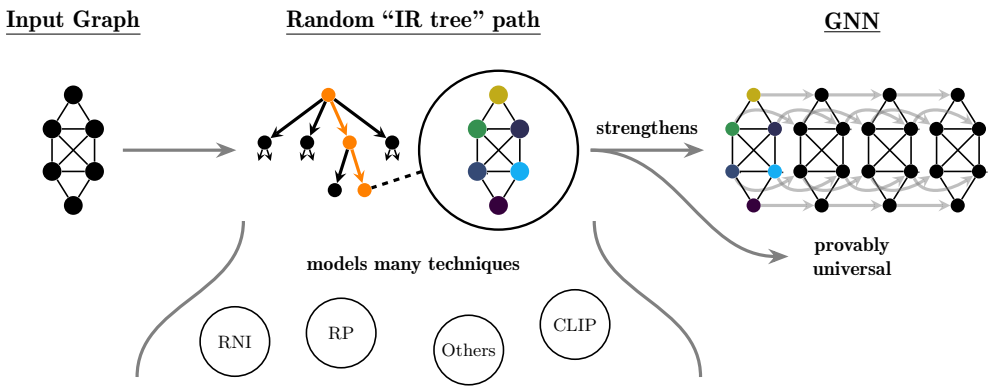

Figure 1: Summary of our new approach: we describe a framework based on "random IR tree paths", called IRNI. Through its parameters, the model is able to exactly model many previous techniques. We prove that any IRNI-enhanced GNN is universal.

context of machine learning on graphs already led to two success stories, namely the efficient Weisfeiler-Leman (WL) Kernels (Shervashidze et al., 2011) based on color refinement (1-WL) and the more theoretical higher-order graph neural networks (Morris et al., 2019) based on higher-dimensional WL.

However, when it comes to practical graph isomorphism, the use of the 2-dimensional WL is already prohibitive. This is not only due to excessive running time but also due to excessive memory consumption. In the world of isomorphism testing, higher-dimensional Weisfeiler-Leman algorithms remain on the theoretical side. In truth, without fail, modern solvers are IR algorithms (McKay, 1981; Junttila & Kaski, 2011; McKay & Piperno, 2014; Anders & Schweitzer, 2021a). They only use color refinement and instead of higher-dimensional WL rather use individualizations to achieve universality. In contrast to higher-dimensional WL, the IR approach is generic, universal, *and* practical. Uncontested for more than 50 years, IR algorithms have the fastest computation times and acceptable memory consumption for graph isomorphism (McKay & Piperno, 2014). In that sense, the IRNI approach we introduce in this paper is the first time in which universal graph isomorphism techniques that are truly used in practice are transferred into a machine learning context. An important consequence of this is that ML practitioners can now readily transfer existing IR-related results to ML on graphs (Sec. 4.3).

2 Related Work

Graphs are a powerful means of representing semantic information. Since graphs are a very general data type and most other data types are special cases, graphs have many applications. They can be used to extend or combine other data types like text, images, or time series (Noble & Cook, 2003; Vazirgiannis et al., 2018) and there are also data sets specific to graphs. Most commonly, these data sets are related to biology or chemistry. However, computer vision, social networks, and synthetic graphs without a related field are also present (Morris et al., 2020a). Neural learning on structured data like graphs was first introduced in (Baskin et al., 1997; Sperduti & Starita, 1997). Recently, a more specific deep learning model was pioneered for graphs, the graph neural network (GNN) (Gori et al., 2005; Scarselli et al., 2009). The GNN led to the development of a multitude of related models (Duvenaud et al., 2015; Li et al., 2016) usually referred to just as GNNs. GNNs allow for the joint training of graph feature extraction and classification, which previous models did not. Gilmer et al. (2017) gave a very general characterization of GNNs called message-passing neural networks (MPNN), which most GNN models can be characterized as. Lately, multiple concepts from other domains of deep learning have been transferred to GNNs like the attention mechanism (Velickovic et al., 2018) and hierarchical pooling (Ying et al., 2018).

Cybenko (1989) proved the first universality result for one of the earliest deep learning models, the smallest possible multilayer perceptron (MLP). This result has since then been expanded to different activation functions (Leshno et al., 1993; Barron, 1994), to width-bounded MLPs (Lu et al., 2017; Liang & Srikant,

2017), and more recently to different layered artificial neural networks like the convolutional neural network (Zhou, 2020). Analogous results had been lacking for MPNNs, which are now well-established to be non-universal (Xu et al., 2019; Morris et al., 2019; Abboud et al., 2021). Following this finding, multiple attempts were made to establish universal ML models on graph data. Murphy et al. (2019) proposed relational pooling (RP), Sato et al. (2021) proposed random node initialization (RNI), and Dasoulas et al. (2020) proposed the colored local iterative procedure (CLIP), all of which provide universality to MPNNs (Abboud et al., 2021) and are closely related to one another as methods using universal random features (URF). Morris et al. (2019; 2020b) proposed k-GNNs based on the k-dimensional Weisfeiler-Leman algorithm and expanded on it by proposing the δ-k-GNN a local approximation variant of the k-GNN. Maron et al. (2019a;b) propose provably powerful graph networks, which are 2-WL powerful, and invariant graph networks, which are proven to be universal, see Morris et al. (2021) for further pointers.

In this paper, we only consider methods that are scalable, practical, and universal at the same time. For a comparison between universal and non-universal approaches we refer to existing literature (Dasoulas et al., 2020; Morris et al., 2019; 2020b; Maron et al., 2019a). As expected non-universal approaches can outperform universal approaches in tasks that do not require high expressivity. However, non-universal approaches fail to achieve high performance on tasks that require high expressivity. Therefore, only methods employing random features that grant universality (Murphy et al., 2019; Dasoulas et al., 2020; Sato et al., 2021), or universal random features (URF), are considered. Some readers might wonder if typical graph data augmentation schemes should also be considered here. Examples would include deleting/adding edges/nodes. While these changes to graphs can induce the capacity to distinguish graphs from one another, we expected that universality of these methods is hard to prove. For deletions, this is essentially due to the various reconstruction conjectures, which so far have no proof. It is currently unknown if the set of all graphs with one node/edge deleted can be used to reconstruct the original graph, which would be required for these operations to be universal. As for node addition, the IRNI framework subsumes this operation as coloring a set of nodes is equivalent to node addition. Lastly, edge addition can be viewed as removing edges from the inverted graph and therefore has the same issues as edge deletion. For these reasons, we do not consider the vast number of data augmentations that are not provably universal (Ding et al., 2022; Papp et al., 2021).

3 Background

Here we cover existing background necessary to understand the IRNI framework. Specifically we define graphs as well as how they are used in ML followed by coloring of graphs. We then cover color refinement a fundamental backbone of individualization refinement, the practical algorithm the IRNI framework is based on. This is followed by a definition of GIN networks the specific MPNN model we use. Lasty, we cover the specifics of RP, RNI, and CLIP, which are the related methods that can be described in the IRNI framework and that we compare later on.

3.1 Graphs and Colorings

We consider undirected, finite graphs $G = (V, E)$ which consist of a set of vertices $V \subseteq \mathbb{N}$ and a set of edges $E \subseteq V^2$, where E is symmetric. From this point onward, let $n := |V|$ and $V = \{1, \dots, n\}$. Additionally, we let $\mathcal{G}$ denote the set of all graphs, while $\mathcal{G}_n$ denotes the set of all graphs on n vertices.

In ML contexts, graphs typically carry a node representation in $\mathbb{R}^d$, which we denote by $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$. IR-tools require these node representations to be discrete. In other contexts, discretization can be difficult and techniques are being actively researched (Morris et al., 2016). However, the discretization is not critical for our purpose since we only require this encoding to compute a tuple of nodes $(w_1, w_2, \dots)$ from an IR algorithm. After that, our approach continues on the original node representation. Let $enc : \mathbb{R}^d \times \mathcal{G} \to \mathbb{N}$ be an arbitrary isomorphism-invariant encoding of the node representations. In practice, it is best to choose an encoding for which $enc(\boldsymbol{x}_v, G) = enc(\boldsymbol{x}_w, G)$ if and only if $\boldsymbol{x}_v = \boldsymbol{x}_w$, however, this is not a requirement for any of the results we present. Elaborating further, if $enc(\boldsymbol{x}_v, G) = enc(\boldsymbol{x}_w, G)$ even though $\boldsymbol{x}_v \neq \boldsymbol{x}_w$ then more nodes might be "individualized" than are strictly necessary, still resulting in a universal and equivariant method as described in Sec. 4.

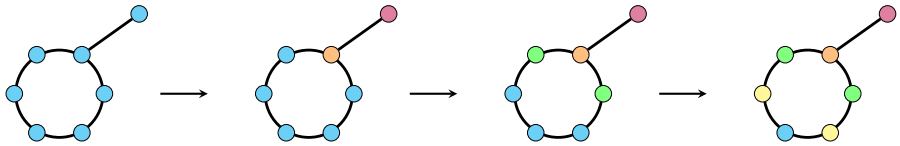

Figure 2: A run of naive refinements on a graph. The sequence of naive refinements ends in the coarsest equitable coloring, which can not be refined further using the naive refinement. Note how each coloring is finer than the previous coloring.

A (node) *coloring* is a surjective map $\pi\colon V \to \{1, \dots, k\}$. We interpret the node representations as colors using *enc*, i.e. $\pi(i) := enc(\boldsymbol{x}_i, G)$. We call $\pi^{-1}(i) \subseteq V$ the i-th cell for $i \in \{1, \dots, k\}$. If $|\pi(V)| = n$ then π is *discrete*. This means every node has a unique color in π. Note that in the following, we always use "discrete" in this sense. Furthermore, we say a coloring π is *finer* than π' if $\pi(v) = \pi(v') \implies \pi'(v) = \pi'(v')$ holds for every $v, v' \in V$. We may also say π' is *coarser* than π. See Figure 2, where each consecutive coloring of the graph is finer than the previous one.

We should remark that the IR machinery generalizes to directed and edge-colored graphs. One example of achieving this is by subdividing each edge with additional colored vertices, where the additional vertices model the edge color or edge direction (see McKay & Piperno for more details). Alternatively, said features can also be added to IR algorithms directly (Piperno, 2018).

We denote by $N_G(v)$ the neighborhood of node v in graph G.

3.2 Color Refinement

We now define color refinement, which all practical graph isomorphism solvers use. A coloring π is *equitable* if for every pair of (not necessarily distinct) colors i, j, the number of j-colored neighbors is the same for all i-colored vertices. Equitable colorings are precisely the colorings for which color refinement cannot be employed to distinguish nodes further. For a colored graph (G, π) there is (up to renaming of colors) a unique coarsest equitable coloring finer than π (McKay, 1981). This is precisely the coloring computed by color refinement.

A more algorithmic way to describe the refinement is to define for a colored graph (G, π) the naively refined graph (G, π^r) where $\pi^r(v) := (\pi(v), \{\!\{\pi(v') \mid v' \in N_G(v)\}\!\})$. The naive refinement r is applied exhaustively, i.e., until vertices cannot be partitioned further. The result is precisely the coarsest equitable coloring. Figure 2 illustrates the color refinement process.

Note that color refinement, and hence coarsest equitable colorings, are strongly related to MPNNs, as is explained below.

3.3 Message Passing Neural Networks

Formally a message passing update can be formulated as:

$$\boldsymbol{x}_{v,t+1} := \text{combine}(\boldsymbol{x}_{v,t}, \text{aggregate}(\{\!\{\boldsymbol{x}_{w,t} | w \in N_G(v)\}\!\})), \tag{1}$$

where G is a graph and $\boldsymbol{x}_{v,t}$ is the vector representation of node v at time t. In this context, time t is usually interpreted as the layer and the aggregate function is typically required to be invariant under isomorphisms. We want to point out the similarity between the refinement r in color refinement and the message passing update (Equation 1). MPNNs are just computing on the colors computed by color refinement, which is why MPNNs are at most as powerful as color refinement in distinguishing graphs. We refer to Xu et al. (2019) and Morris et al. (2019) for a formal comparison. Common instances of MPNNs are graph convolutional networks (Duvenaud et al., 2015), graph attention networks (Velickovic et al., 2018), and graph isomorphism networks (GIN) (Xu et al., 2019). We use GIN throughout this paper, arguably the most efficient of these models (Dasoulas et al., 2020).

A GIN is characterized by being simple and yet as powerful as the color refinement algorithm. Given an arbitrary multi-layer perceptron $\mathrm{MLP}^{(k)}$ in layer k, $\epsilon^{(k)}$ a learnable parameter of layer k, and $h_v^{(0)}$ the input representation of node v. GIN updates its node representations $h_v^{(k)}$ in layer k as follows:

$$h_v^{(k)} := \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)h_v^{(k-1)} + \sum_{u\in\mathcal{N}_G(v)} h_u^{(k-1)}\right). \tag{2}$$

3.4 Random Features

Intuitively, using random features is a process that introduces randomized information into the input before processing it using machine learning techniques. For example, one might delete a random node or attach random numbers to feature vectors. We do not require a more formal notion since, in this paper, we are only interested in universal random features (URF). We use URF to refer to methods that provably enable their universality if used in conjunction with MPNNs. We next summarize all existing URF techniques.

RP attaches a random permutation to the graph by attaching to each node a one-hot encoding of its image. To understand RNI, let G be a graph with node representations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $d \in \mathbb{N}$ a constant, random node initalization (RNI) concatenates d features sampled from a random distribution $\mathcal{X}$ to each node $\forall v \in \{1, \ldots, n\} : \boldsymbol{x}_v \leftarrow \text{concatenate}(\boldsymbol{x}_v, r_1, \ldots, r_d), r_1 \ldots, r_d \sim \mathcal{X}$. CLIP first apply color refinement and then individualize each node of each color class by assigning a one-hot encoding of a natural number to it. If C is the set of all nodes of one color class, then each node $v \in C$ is randomly assigned a unique number in $\{0, \ldots, |C|-1\}$, which is then one-hot encoded and concatenated onto its node representation $\boldsymbol{x}_v$.

4 Random Features from Individualization Refinement

First, we describe individualization-refinement trees followed by the individualization-refinement node initialization (IRNI) framework. Then, we demonstrate that RNI, CLIP, and RP can be expressed as a manifestation of this framework. Lastly, we give several theoretical insights into the framework and prove a general universality for these methods under a natural compatibility constraint.

4.1 Individualization Refinement Trees

Individualization refinement (IR) trees are the backtracking trees of practical graph isomorphism algorithms. We use randomly sampled leaves of these trees as random features. These leaves correspond to sequences of nodes $(w_1, \ldots, w_k)$ of the graph, which we translate into features of the MPNN. One central property of the sequence is that distinguishing its nodes from other nodes in the graph and applying color refinement yields a discrete coloring.

In the following, we describe all the necessary ingredients of IR for the purposes of this paper. The IR paradigm is a complex machinery refined over many decades into sophisticated software libraries. We refer to McKay & Piperno (2014) and Anders & Schweitzer (2021a) for an exhaustive description.

Refinement. The most crucial subroutine of an IR algorithm is the *refinement* $\mathrm{Ref} : G \times \Pi \times V^* \to \Pi$, where Π denotes the set of all vertex colorings of G and V^* denotes a string of vertices (* is the Kleene star). A refinement must satisfy two properties: it must be invariant under isomorphism and individualize vertices in $\nu \in V^*$, i.e., let $\pi' = \mathrm{Ref}(G, \pi, \nu)$, then for all $v \in \nu$ it holds that ${\pi'}^{-1}(\pi'(v)) = \{v\}$. Our definition of refinement is slightly more general compared to McKay & Piperno (2014), which leads to a slight technicality that we discuss in the appendix.

In practice, IR tools use color refinement as their refinement (see Sec. 3.2). We denote color refinement as $\mathrm{CR}(G, \pi, \epsilon)$, where ϵ denotes the empty sequence (explained further below).

Individualization. IR algorithms make use of *individualization*, a process that artificially forces a node into its own color class, distinguishing it. To record which vertices have been individualized we use a sequence

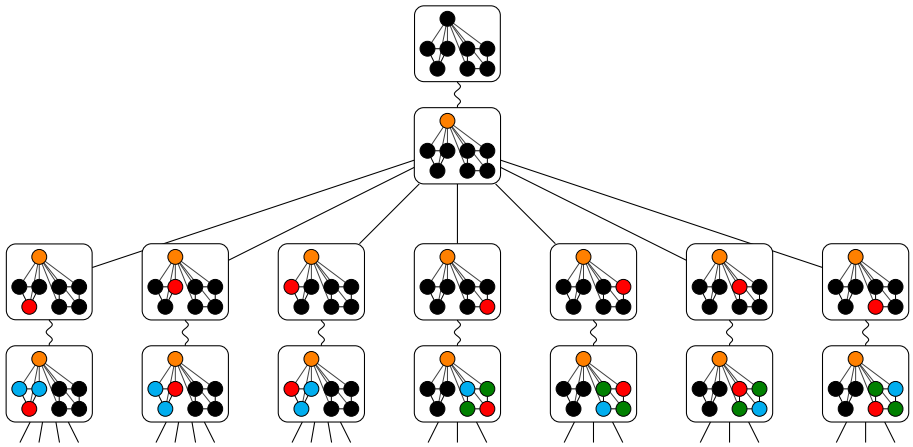

Figure 3: In IR, refinement and individualization are alternatingly applied. This continues until the coloring of the graph becomes discrete. Two nodes connected by a squiggly line are considered one node of the IR tree: they illustrate the coloring of the graph before and after color refinement.

$\nu = (v_1, \ldots, v_k) \in V^*$. We modify color refinement so that $\mathrm{CR}(G, \pi, \nu)$ is the unique coarsest equitable coloring finer than π in which every node in ν is a singleton with its own artificial color. Artificial distinctions caused by individualizations are thus taken into account.

Cell selector. In a backtracking fashion, the goal of an IR algorithm is to reach a discrete coloring using color refinement and individualization. For this, color refinement is first applied. If this does not yield a discrete coloring, individualization is applied, branching over all vertices in one non-singleton cell. The task of the *cell selector* is to (isomorphism-invariantly) pick the non-singleton cell. Figure 3 illustrates this process. While many choices within certain restrictions are possible, one example that we will also use later on is the selector that always chooses the first, largest non-singleton cell of π.

IR trees. We first give a formal definition of the IR tree $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$, followed by a more intuitive explanation. Nodes of $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$ are sequences of vertices of G and the root is the empty sequence $\epsilon = ()$. If $\nu = (v_1, \ldots, v_k)$ is a node in $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$ and $C = \mathrm{Sel}(G, \mathrm{Ref}(G, \pi, \nu))$ is the selected cell, then the set of children of ν is $\{(v_1, \ldots, v_k, v) \mid v \in C\}$, i.e., all extensions of ν by one node v of the selected cell C. The root represents the graph (with no individualizations) after refinement (see Figure 3). A node ν represents the graph after all nodes in ν have been individualized followed by refinement (see Figure 3). A root-to-ν walk of the tree is naturally identified with a sequence of individualizations in the graph: in each step i of the walk, one more node v_i belonging to a non-trivial color class C is individualized (followed by refinement). The sequence of individualizations $(v_1, \ldots, v_k)$ uniquely determines the node of the IR tree in which this walk ends. This is why we identify the name of the node with the sequence of individualizations necessary to reach the node: the sequence of individualizations necessary to reach ν is $(v_1, \ldots, v_k) = \nu$. $\Gamma_{\mathrm{Ref,Sel}}(G, \pi, \nu)$ denotes the subtree of $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$ rooted in ν. We remark that the notation $(G, \pi)^\varphi$ simply means we apply φ to G and π, as noted in the lemma below. Isomorphism invariance of the IR tree follows from isomorphism invariance of Sel and Ref:

Lemma 1 (McKay & Piperno (2014))**.** *Let $\varphi\colon V \to V$ denote an* automorphism *of (G, π), i.e., $(G, \pi)^\varphi = (\varphi(V), \varphi(E), \varphi(\pi)) = (V, E, \pi) = (G, \pi)$. Let $\mathrm{Aut}(G, \pi)$ denote all automorphisms of (G, π). Then, if ν is a node of $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$ and $\varphi \in \mathrm{Aut}(G, \pi)$, then ν^φ is a node of $\Gamma_{\mathrm{Ref,Sel}}(G, \pi)$ and $\Gamma_{\mathrm{Ref,Sel}}(G, \pi, \nu)^\varphi = \Gamma_{\mathrm{Ref,Sel}}(G, \pi, \nu^\varphi)$.*

Generally, we refer to a process or object, as *isomorphism-invariant*, whenever it produces the same result for isomorphic inputs.

We want to remark again that leaves of an IR tree correspond to discrete colorings of a graph. In fact, the set of all leaves of an IR tree forms a complete isomorphism invariant: two isomorphic graphs will have the same set of leaves, whereas two non-isomorphic graphs are guaranteed to have a distinct set of leaves (see Lemma 4 of McKay & Piperno (2014)).

Random IR walks. There are various ways to traverse and use IR trees. Traditionally, solvers (e.g., NAUTY) solely used deterministic strategies, such as depth-first traversal (McKay & Piperno, 2014). Only recently competitive strategies based solely on random traversal, i.e., DEJAVU (Anders & Schweitzer, 2021a), have emerged (see Section 4.3). We make use of this recent development, by using *random root-to-leaf walks* of the IR tree $\Gamma_{\text{Ref,Sel}}(G, \pi)$. We begin such a walk in the root node of $\Gamma_{\text{Ref,Sel}}(G, \pi)$. We repeatedly choose uniformly at random a child of the current node until we reach a leaf ν of the tree. Then, we return the leaf $\nu = (w_1, \ldots, w_k)$. A crucial property is that since $\Gamma_{\text{Ref,Sel}}(G, \pi)$ is isomorphism-invariant (Lemma 1), random walks of $\Gamma_{\text{Ref,Sel}}(G, \pi)$ are isomorphism-invariant as well:

Lemma 2 (Anders & Schweitzer (2021a))**.** *As a random variable, the graph colored with the coloring of the leaf resulting from a random IR walk is isomorphism-invariant.*

Lemma 2 is also true when restricting random walks to prefixes of a certain length d. We stress that random IR walks are conceptually unrelated to random walks in the graph itself considered elsewhere (Nikolentzos & Vazirgiannis, 2020). Our next step is to insert the sequence of nodes defined by random IR walks into MPNNs.

4.2 Individualization Refinement Node Initalization

Let G be a graph with node representations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $d \in \mathbb{N}$ a constant. Individualization-refinement node initalization (IRNI) computes a random IR walk $w = (w_1, \ldots, w_k)$ in $\Gamma_{\text{Ref,Sel}}(G, \pi)$, where $\pi(i) := enc(x_i, G)$. If $k \geq d$, we take a prefix of length d, i.e., $(w_1, \ldots, w_d)$, providing d nodes to be individualized. IRNI then concatenates d features that are either 0 or 1 depending on this prefix: We set $\forall v \in \{1, \ldots, n\}$: $\boldsymbol{x}_v \leftarrow \boldsymbol{x}_v \circ (\mathbb{1}_{w_1=v}, \ldots, \mathbb{1}_{w_d=v})$, which means that the j-th feature of node v is set to 1 if v is the j-th node that was individualized (i.e., $w_j = v$) and 0 otherwise. This guarantees that node v is individualized if and only if it appears in the prefix. If $k < d$, then we simply "fill up" the walk with nodes in an isomorphism-invariant manner using the discrete coloring $\text{Ref}(G, \pi, (w_1, \ldots, w_k))$: we add nodes in order of their color, first the node with the smallest color, then with the second smallest color, and so forth. We abbreviate IRNI with constant d as d-IRNI.

Due to the dependence on an underlying IR-tree, both the refinement Ref and cell selector Sel are natural hyperparameters of IRNI. Unless stated otherwise, we assume that an arbitrary refinement and an arbitrary cell selector are used. If we want to state a specific refinement or cell selector, we do so using d-IRNI(Ref) and d-IRNI(Ref, Sel), respectively.

IRNI depends on the random walk in the IR tree and we will prove its universality in Theorem 4. Thus, IRNI is a URF (analogous to RNI, CLIP, and RP). This justifies ensembling over this randomness, which we will refer to as ensembling over randomness (EoR). Specifically, we average the predictions of an MPNN over some URF.

We now define some specific instances of IRNI by applying different refinements. Let the trivial refinement $\text{TR}(G, \pi, \nu)$ only individualize the vertices ν in π, followed by no further refinement. A random walk of $\Gamma_{\text{TR}}(G, \pi)$ thus picks a random permutation of vertices that respects only the initial color classes of π (i.e., the first vertex will always be of the first selected color of π, and so forth). In case of uncolored graphs (G, π) where π is the trivial coloring, random walks truly only become random permutations of vertices of G. In this case, it follows that for $G \in \mathcal{G}_n$ n-IRNI(TR) is equivalent to RP. However, we can enforce this even for colored graphs by actively ignoring the colors of π, resulting in what we call the *oblivious refinement* $\text{OR}(G, \pi, \nu) := \text{TR}(G, V(G) \mapsto 1, \nu)$. In this case, random walks are always random permutations of vertices of G. Hence, it follows that for $G \in \mathcal{G}_n$ n-IRNI(OR) is RP. We remark that the only difference between RNI and RP is in the encoding of the individualizations.

Based on TR and CR, we define $\mathrm{CTR}(G, \pi, \nu) := \mathrm{TR}(G, \mathrm{CR}(G, \pi, \epsilon), \nu)$. Note that CTR applies color refinement to the graph, followed by trivial refinement of nodes in the resulting color classes. By definition, for $G \in \mathcal{G}_n$ n-IRNI(CTR) is thus an alternative description of CLIP.

4.3 IR algorithms and IRNI

We now discuss the relationship between IR algorithms and MPNNs using IRNI. First, we remark that in terms of solving graph isomorphism, the use of repeated random IR walks has recently been proven to be a near-optimal traversal strategy of IR trees (Anders & Schweitzer, 2021b), where near-optimal refers to a logarithmic gap between the lower and upper bound. This is in contrast to deterministic traversal strategies such as depth-first search or breadth-first search, which have a quadratic overhead (Anders & Schweitzer, 2021b). URF are thus closely related to these optimal strategies. The ensembling defined in the previous section even seems to mimic the way the aforementioned near-optimal IR algorithm operates (Anders & Schweitzer, 2021b). In fact, the currently fastest practical graph isomorphism algorithm DEJAVU (Anders & Schweitzer, 2021a) uses essentially the same strategy. Moreover, the use of random IR walks has additional inherent benefits, such as "implicit automorphism pruning", i.e., the automatic exploitation of symmetry in the input (Anders & Schweitzer, 2021a). This translates to MPNNs with IRNI, in that if individualizations across multiple random walks are made on nodes that are symmetrical to each other, this does not introduce any additional randomness: due to their isomorphism-invariance (or equivariance), symmetrical nodes are indistinguishable by MPNNs by design.

Previously, we discussed that a crucial property of MPNNs is that their result is isomorphism-invariant, i.e., it only depends on the isomorphism type. This is not true in the deterministic sense for IRNI. However, because of Lemma 2, the result only depends on the isomorphism type and the randomness.

Lemma 3. *Let* Ref *be any refinement. Let f be the function computed by d-*IRNI(Ref)*, mapping a graph G and a random seed $s \in \Omega$ from the sample space of random IR walks Ω to a value $f(G, s) \in \mathbb{R}$. Then for every permutation φ, we have that the random variables $s \mapsto f(G, s)$ and $s \mapsto f(\varphi(G), s)$ have the same distribution.*

Proof. The result follows directly from the isomorphism-invariance of MPNNs and Lemma 2. Nevertheless, we give a more extensive proof here.

First, we recall that MPNNs are isomorphism-invariant (or equivariant): for every permutation φ, by definition, a MPNN produces the same result for G as for $\varphi(G)$. Hence, we only need to proof that the augmentations made through random IR walks satisfy the claim.

We note that the IR tree of G is $\Gamma_{\mathrm{Ref}}(G)$ and the IR tree of $\varphi(G)$ is $\Gamma_{\mathrm{Ref}}(\varphi(G)) = \varphi(\Gamma_{\mathrm{Ref}}(G))$ (see Lemma 1 in McKay & Piperno (2014)). Intuitively, this means that ignoring the permutation φ we are drawing randomly from the same distribution.

Hence, randomly sampling walks from these trees will result in the same nodes, except for applying the permutation φ: if $\nu \in \Gamma_{\mathrm{Ref}}(G)$, then $\varphi(\nu) \in \Gamma_{\mathrm{Ref}}(\varphi(G))$. We remark that G individualized with ν is isomorphic to $\varphi(G)$ individualized with $\varphi(\nu)$. This corresponds to the augmentation made to the MPNN in IRNI. Hence, overall, it follows that the MPNN must give the same result for G augmented with ν, as well as $\varphi(G)$ augmented with $\varphi(\nu)$. □

We now provide a universality theorem for IRNI, in a similar fashion as Abboud et al. (2021) does for RNI. Let us first recall a definition of Abboud et al. (2021): let $\mathcal{G}_n$ denote the class of all n-vertex graphs and $f \colon \mathcal{G}_n \to \mathbb{R}$. We say that some randomized function $\mathcal{X}$ that associates with a graph $G \in \mathcal{G}_n$ a random variable $\mathcal{X}(G)$ is an (ϵ, δ)-approximation of f if for all $G \in \mathcal{G}_n$ it holds that $\Pr(|f(G) - \mathcal{X}(G)| \leq \epsilon) \geq 1 - \delta$.

The theorem proven by Abboud et al. (2021) is based on the fact that RNI fully individualizes a graph with high probability, and all fully individualized representations of a graph together constitute a complete isomorphism invariant. The crucial insight we exploit is that IR trees constitute a complete isomorphism invariant as well, specifically, even the set of all leaves of an IR tree suffices. Since the power of MPNNs is

limited by color refinement, the only additional requirement needed is that the refinement used for random walks must be at most as powerful as color refinement.

Theorem 4. *Let* Ref *be a refinement that computes colorings coarser or equal to* CR*. Let* $n \geq 1$ *and let* $f \colon \mathcal{G}_n \to \mathbb{R}$ *be invariant. Then, for all* $\epsilon, \delta > 0$*, there is an MPNN with* $(n-1)$-IRNI(Ref) *that* (ϵ, δ)*-approximates* f*.*

Proof. We prove the theorem using a combination of Theorem 2 from (Morris et al., 2019), the universality result of RNIs given in (Abboud et al., 2021), and the basic definition of IR trees. Since graphs have n nodes all possible random IR walks considered by $(n-1)$-IRNI(Ref) are random IR walks ending in a leaf node of $\Gamma_{\text{Ref}}(G, \pi)$ (see Section 4.1). If we were to individualize the sequence of nodes $(w_1, \ldots, w_k)$ corresponding to a leaf and apply the refinement Ref, the coloring of the entire graph would become discrete. By assumption, color refinement always produces colorings finer or equal to Ref, so applying color refinement also produces a discrete coloring.

By the definition of $(n-1)$-IRNI(Ref), the nodes contained in $\{w_1, \ldots, w_k\}$ all have distinct features not shared by any of the other nodes in the graph. This means that the nodes in $\{w_1, \ldots, w_k\}$ are indeed initially individualized in the MPNN. Now, it is well-known that Theorem 2 of Morris et al. (2019) (see also (Xu et al., 2019)) shows that there is an MPNN that produces the same partitioning of colors that color refinement would, i.e., in our case, yields a discrete partitioning of vertices. In other words, we can assume that the graph is individualized.

This suffices to apply the universality result of Abboud et al. (2021) (see Lemma A2, Lemma A3, and Lemma A4 in (Abboud et al., 2021), which build upon (Barceló et al., 2020)), which solely depends on individualizing the graph. In particular, the proof of Abboud et al. (2021) builds a $\mathcal{C}^2$ sentence which identifies discretely colored graphs (Lemma A3). In turn, a disjunction identifying any possible discretely colored graph for a given graph is constructed (Lemma A4). Since the set of leaves of an IR tree are a complete isomorphism invariant, it suffices to build this disjunction over only those discretely colored graphs that correspond to a leaf in the IR tree. □

The hyperparameters of IR open up more opportunities to transfer results into the realm of MPNNs. We give one such example. We argue that with a specific cell selector, 3-connected planar graphs can be detected with 4-IRNI(CR).

Theorem 5. *Let* $\mathcal{P}_n$ *denote the class of 3-connected planar graphs. Let* $n \geq 1$ *and let* $f \colon \mathcal{P}_n \to \mathbb{R}$ *be invariant. Then, for all* $\epsilon, \delta > 0$*, there is a cell selector* Sel *(which does not depend on* n*) and an MPNN with* 4-IRNI(CR, Sel) *that* (ϵ, δ)*-approximates* f*.*

Proof. First of all, we argue that individualizing a node of degree 5 and three of its neighbors surely suffices to make the graph discrete: this follows from Lemma 22 of Kiefer et al. (2017), which proves that individualizing 3 vertices on any common face followed by color refinement suffices to make the coloring discrete. Note that individualizing a node of degree 5 and three of its neighbors surely individualizes three vertices on a common face. Using the arguments from the proof of Theorem 4 again, we can see that this would indeed suffice to show the claim.

It remains to be shown that there is a cell selection strategy achieving the above. In the first step, the cell selector chooses an (isomorphism-invariant) color class consisting of degree 5 vertices. We remark that, due to Euler's formula, the average degree of a planar graph is less than 6, so such a node always exists. In the next step, we choose a non-trivial class containing only neighbors of the individualized degree 5 node v. We argue that unless all neighbors of v have been individualized, there is a non-trivial color class consisting of neighbors of v. Indeed, if there is a non-trivial class containing neighbors of v, the class may only contain such neighbors since color refinement distinguishes neighbors of v from non-neighbors of v. Here we use that v is individualized. We repeat the step of choosing a non-singleton class of neighbors of v and individualizing a node within it. If at any point no non-trivial class of neighbors exists, we are done: this means that the neighbors are fully discrete. This in turn suffices to show the claim. □

Method	PROTEINS	MUTAG	NCI1	TRI	TRIX	EXP	CEXP	CSL
None	0.68 ± 0.06	0.89 ± 0.06	0.81 ± 0.02	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.01	0.74 ± 0.02	0.50 ± 0.00
RNI	0.66 ± 0.02	**0.89 ± 0.04**	**0.81 ± 0.01**	**0.99 ± 0.01**	**0.99 ± 0.00**	**0.97 ± 0.03**	0.95 ± 0.10	0.85 ± 0.06
CLIP	0.65 ± 0.05	**0.85 ± 0.09**	0.81 ± 0.01	**0.99 ± 0.00**	0.81 ± 0.05	**0.99 ± 0.04**	**0.99 ± 0.02**	**1.00 ± 0.01**
RP	**0.74 ± 0.04**	**0.86 ± 0.07**	**0.81 ± 0.01**	**0.99 ± 0.00**	0.82 ± 0.03	**0.96 ± 0.02**	0.97 ± 0.02	**1.00 ± 0.00**
IRNI(CR)	**0.75 ± 0.04**	**0.85 ± 0.05**	**0.82 ± 0.02**	**0.99 ± 0.01**	0.73 ± 0.04	**0.99 ± 0.04**	0.95 ± 0.14	**1.00 ± 0.00**
RNI_{EoR}	0.69 ± 0.05	**0.94 ± 0.03**	**0.85 ± 0.02**	**1.00 ± 0.00**	**1.00 ± 0.00**	**0.99 ± 0.01**	**0.98 ± 0.05**	0.93 ± 0.06
CLIP_{EoR}	0.67 ± 0.03	**0.92 ± 0.05**	0.82 ± 0.02	**1.00 ± 0.00**	0.95 ± 0.05	**1.00 ± 0.00**	**0.97 ± 0.08**	**1.00 ± 0.00**
RP_{EoR}	**0.78 ± 0.04**	0.84 ± 0.12	**0.87 ± 0.02**	**1.00 ± 0.00**	0.95 ± 0.05	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.01**
IRNI(CR)_{EoR}	0.74 ± 0.04	0.87 ± 0.08	0.82 ± 0.02	**1.00 ± 0.00**	0.94 ± 0.05	**0.99 ± 0.02**	**0.97 ± 0.06**	**0.99 ± 0.02**
SOTA_{URF}	0.81 ± 0.03	0.95 ± 0.03	0.88 ± 0.01	0.91 ± NA	0.93 ± NA	NA	NA	NA
SOTA^*_{URF}	0.77 ± 0.04	0.94 ± 0.04	NA	NA	NA	0.98 ± 0.02	NA	0.91 ± 0.07

Table 1: The AUROC of a GIN network with RNI, CLIP, RP, IRNI(CR), and without any (None) of these on selected data sets. $_{EoR}$ indicates the use of ensembling over randomness. Bold entries indicate statistically significant best values, for this EoR is treated as separate from no EoR. SOTA_{URF} indicates the state-of-the-art AUROC. SOTA^*_{URF} indicates the state-of-the-art accuracies, note that these are not directly comparable. Specifically, SOTA_{URF} and SOTA^*_{URF} refer to the results from the previous evaluations in (Murphy et al., 2019; Sato et al., 2021; Dasoulas et al., 2020; Abboud et al., 2021), from which we always chose the best value.

More results of this kind can be shown. For example, it is known that strongly regular graphs require at most $O(\sqrt{n}\log n)$ individualizations (Babai, 1980). In fact, for all but an exponentially small fraction of graphs, d-IRNI(CR) with small d suffices.

Theorem 6. *There is an absolute constant $c > 1$ such that the following holds. Let $n \geq 1$ and $d \in \mathbb{N}$, then there is a graph class $\mathcal{G}'_n$ containing all but at most a $1/c^{dn}$ fraction of all graphs for which the following holds. Let $f\colon \mathcal{G}'_n \to \mathbb{R}$ be invariant, then, for all $\epsilon, \delta > 0$, there is an MPNN with d-IRNI(CR) that (ϵ, δ)-approximates f.*

Proof. To prove the theorem, we use the same technique as before. We only need to observe that for most graphs, after color refinement is applied, d arbitrary individualizations in non-singleton cells cause discretization of the graph. This, however, is a classic theorem by Babai & Kucera (1979, Theorem 4.1) showing the fraction of graphs for which this fails is at most $1/c^{dn}$. □

5 Experiments

We compare the URF schemes RNI, CLIP, RP, and IRNI(CR) and verify their increase in expressivity. We do so by applying them to synthetically crafted, hard data sets as well as standard practical data sets. Furthermore, we propose an automated training approach used throughout the benchmarks.

Network architectures and optimization. For all experiments, we use the same general architecture, the Adam optimizer, and use the area under the receiver operating characteristic (AUROC). We optimize each method using a bayesian hyperparameter search in the same hyperparameter space. To estimate the performance, we use Monte Carlo cross-validation in an outer test loop and an inner validation loop estimating nested 10 × 9-fold cross-validation. The bayesian hyperparameter search is capped at evaluating 50 points in hyperparameter space. To encourage the models to optimize faster as well as to avoid overfitting, we add a penalty to the AUROC estimate based on some hyperparameters. The reported test AUROC does not include these penalties. To compute the node sequence for IRNI(CR) as well as the color refinement for CLIP we use DEJAVU (Anders & Schweitzer, 2021a). These choices are specified further in the appendix. In the following, we refer to a GIN without any URF as None, while we refer to a GIN with some initialization as RNI, CLIP, RP, or IRNI(CR) depending on the URF that is used. Each of these methods is also limited in the number of dimensions added.

Data sets. We evaluate different models on datasets used in prior work on URF, specifically EXP, CEXP, TRI, TRIX, CSL, PROTEINS, MUTAG, and NCI1 (Srinivasan et al.; Borgwardt et al., 2005; Wale & Karypis, 2006; Murphy et al., 2019; Sato et al., 2021; Abboud et al., 2021). EXP, CEXP, TRI, TRIX, and CSL are synthetic data sets made up of graphs not distinguishable by the color refinement algorithm. TRI and TRIX contain 3-regular graphs and use the same training set while differing in the test set. The task is to detect triangles. EXP and CEXP consist of graphs carefully constructed so that each graph is in a pair that is indistinguishable by color refinement while encoding a satisfiable and unsatisfiable formula respectively. For CEXP 50% of all satisfiable graphs are modified to be distinguishable by color refinement from their unsatisfiable counterparts. CSL consists of 41-cycles with regular skip-connections according to 10 co-primes of 41. Each co-prime defines one class for the CSL task. The results of this experiment can be found in Table 1.

6 Discussion

We notice that on the synthetic hard datasets TRI, TRIX, EXP, CEXP, and CSL, all methods improve the discriminatory power compared to not using any form of URF. We now compare the methods based on the three primary parameters encoding, ensembling, and refinement.

Concerning the encoding on TRIX and CSL, there seem to be noticeable differences between RNI and the other methods. The poorer performance of CLIP, RP, and IRNI(CR) on TRIX is easily explained. Since the task is to detect triangles locally in the graph and the graph is regular, an individualization is required close to the triangle to be able to detect it. RNI individualizes everywhere simultaneously, while CLIP, RP, and IRNI(CR) only individualize locally. This means the detection of triangles depends on random chance. EoR helps since it increases this chance. The difference on CSL is not so clearly explainable. We do not know why RNI performs significantly worse here. Looking more specifically at the difference between RP and RNI, we remark that RP substantially outperforms RNI on PROTEINS, while RNI outperforms RP on MUTAG.

EoR appears almost always to improve the performance. As such, we would advise its consideration whenever one of these URF is used in practice. Notice that its use does not increase training time and only linearly increases prediction time. EoR also appears to guarantee that at least one of RNI, CLIP, RP, or IRNI(CR) will outperform models without this expressibility increase.

Considering the methods that use additional refinement to reduce the introduced randomness, namely CLIP and IRNI(CR), we observe they outperform RP on MUTAG, while RP outperforms the other two on PROTEINS and NCI1. On the synthetic hard datasets, in particular with ensembling, the three methods perform very similarly.

Overall, no method appears to be the uniformly best method for practical use. We suspect that overfitting to the different features introduced by RNI, CLIP, RP, and IRNI(CR) plays a significant role in which of these URF performs best.

7 Conclusion

We introduced IR as it applies to machine learning in the form of the IRNI framework. This enables the development of many URF based on selecting refinement, cell selector, and how to encode individualizations into the network. No URF introduced so far is the clear front runner. However, the BHO tuning approach presented here feasibly allows for the optimization of the IRNI hyperparameters in addition to other model hyperparameters. The IRNI hyperparameters also serve the unifying IRNI umbrella, as we were able to describe all existing and new URF based on these. Moreover, IRNI has a rigorous theoretical foundation ensuring equivariance and universality. We hope this will aid in systematic improvements in future research regarding GNNs.

References

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2112–2118. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/291. URL https://doi.org/10.24963/ijcai.2021/291. Main Track.

Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In Martin Farach-Colton and Sabine Storandt (eds.), *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pp. 73–84. SIAM, 2021a. doi: 10.1137/1.9781611976472.6. URL https://doi.org/10.1137/1.9781611976472.6.

Markus Anders and Pascal Schweitzer. Search Problems in Trees with Symmetries: Near Optimal Traversal Strategies for Individualization-Refinement Algorithms. In Nikhil Bansal, Emanuela Merelli, and James Worrell (eds.), *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 16:1–16:21, Dagstuhl, Germany, 2021b. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-195-5. doi: 10.4230/LIPIcs.ICALP.2021.16. URL https://drops.dagstuhl.de/opus/volltexte/2021/14085%7D.

László Babai. On the complexity of canonical labeling of strongly regular graphs. *SIAM J. Comput.*, 9(1): 212–216, 1980. doi: 10.1137/0209018. URL https://doi.org/10.1137/0209018.

László Babai and Ludek Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pp. 39–46. IEEE Computer Society, 1979. doi: 10.1109/SFCS.1979.8. URL https://doi.org/10.1109/SFCS.1979.8.

Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=r1lZ7AEKvB.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.

Igor I Baskin, Vladimir A Palyulin, and Nikolai S Zefirov. A neural device for searching direct correlations between structures and properties of chemical compounds. *Journal of chemical information and computer sciences*, 37(4):715–721, 1997.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 2126–2132. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/294. URL https://doi.org/10.24963/ijcai.2020/294. Main track.

Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235*, 2022.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/gilmer17a.html.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017. URL http://sites.computer.org/debull/A17sept/p52.pdf.

Tommi A. Junttila and Petteri Kaski. Conflict propagation and component recursion for canonical labeling. In Alberto Marchetti-Spaccamela and Michael Segal (eds.), *Theory and Practice of Algorithms in (Computer) Systems - First International ICST Conference, TAPAS 2011, Rome, Italy, April 18-20, 2011. Proceedings*, volume 6595 of *Lecture Notes in Computer Science*, pp. 151–162. Springer, 2011. doi: 10.1007/978-3-642-19754-3_16. URL https://doi.org/10.1007/978-3-642-19754-3_16.

Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The weisfeiler-leman dimension of planar graphs is at most 3. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pp. 1–12. IEEE Computer Society, 2017. doi: 10.1109/LICS.2017.8005107. URL https://doi.org/10.1109/LICS.2017.8005107.

Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL https://www.sciencedirect.com/science/article/pii/S0893608005801315.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05493.

Shiyu Liang and R. Srikant. Why deep neural networks for function approximation? In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SkpSlKIel.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6232–6240, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper/2019/file/bb04af0f7ecaee4aae62035497da1387-Paper.pdf.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=Syx72jC9tm.

Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pp. 45–87, 1981.

Brendan D. McKay and Adolfo Piperno. Nauty and traces user guide. https://cs.anu.edu.au/people/Brendan.McKay/nauty/nug25.pdf.

Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *J. Symb. Comput.*, 60:94–112, January 2014. ISSN 0747-7171. doi: 10.1016/j.jsc.2013.09.003. URL https://doi.org/10.1016/j.jsc.2013.09.003.

Christopher Morris, Nils M. Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1095–1100, 2016. doi: 10.1109/ICDM.2016.0142.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4602–4609, Jul. 2019. doi: 10.1609/aaai.v33i01.33014602. URL https://ojs.aaai.org/index.php/AAAI/article/view/4384.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020a. URL https://arxiv.org/abs/2007.08663.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21824–21840. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper/2020/file/f81dee42585b3814de199b2e88757f5c-Paper.pdf.

Christopher Morris, Matthias Fey, and Nils Kriege. The power of the weisfeiler-leman algorithm for machine learning with graphs. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4543–4550. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/618. URL https://doi.org/10.24963/ijcai.2021/618. Survey Track.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. In Kirk Pruhs and Christian Sohler (eds.), *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pp. 60:1–60:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.ESA.2017.60. URL https://doi.org/10.4230/LIPIcs.ESA.2017.60.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016. doi: 10.1109/JPROC.2015.2483592.

Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/ba95d78a7c942571185308775a97a3a0-Abstract.html.

Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pp. 631–636, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137370. doi: 10.1145/956750.956831. URL https://doi.org/10.1145/956750.956831.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:21997–22009, 2021.

Adolfo Piperno. Isomorphism test for digraphs with weighted edges. In Gianlorenzo D'Angelo (ed.), *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*,

volume 103 of *LIPIcs*, pp. 30:1–30:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPIcs.SEA.2018.30. URL https://doi.org/10.4230/LIPIcs.SEA.2018.30.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pp. 2847–2854. PMLR, 2017.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. *Random Features Strengthen Graph Neural Networks*, pp. 333–341. 2021. doi: 10.1137/1.9781611976700.38. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976700.38.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. URL http://dl.acm.org/citation.cfm?id=2078187.

Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. Session-based social recommendation via dynamic graph attention networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, pp. 555–563, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3290989. URL https://doi.org/10.1145/3289600.3290989.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. doi: 10.1109/72.572108.

A Srinivasan, S Muggleton, M Sternberg, and R King. Theories for mutagenicity: a study of first-order and feature-based induction. *Artificial Intelligence*, 85(1):2.

Bowen Tang, Skyler T Kramer, Meijuan Fang, Yingkun Qiu, Zhen Wu, and Dong Xu. A self-attention based message passing neural network for predicting molecular lipophilicity and aqueous solubility. *Journal of Cheminformatics*, 12(1):1–9, 2020.

Michalis Vazirgiannis, Fragkiskos D. Malliaros, and Giannis Nikolentzos. GraphRep: Boosting text mining, NLP and information retrieval with graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pp. 2295–2296, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi: 10.1145/3269206.3274273. URL https://doi.org/10.1145/3269206.3274273.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Nikil Wale and George Karypis. Acyclic subgraph based descriptor spaces for chemical compound retrieval and classification. Technical report, University of Minnesota, 2006.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc.

Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 48(2):787–794, 2020. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2019.06.004. URL `https://www.sciencedirect.com/science/article/pii/S1063520318302045`.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. URL `https://www.sciencedirect.com/science/article/pii/S2666651021000012`.

A Refinement Definition

We discuss the slight technicality in our definition of refinement in the individualization-refinement paradigm. Compared to McKay & Piperno (2014), we are lacking the requirement that $\pi' = \mathrm{Ref}(G, \pi, \nu)$ must be finer than π, making our definition of refinements slightly more general. With respect to the results, this implies that for (non-trivially) colored graphs, discrete refined colorings might not respect the initial coloring. This can potentially lead to issues with automorphisms and canonization since, essentially, the isomorphism invariant implied by the refinement is too weak. However, there is a simple fix using other components of the IR framework: we make use of invariants, which are not immediately relevant for IRNI itself. Whenever a coloring π' of a node ν in the tree is discrete, we can use the complete isomorphism invariant $(G^{\pi'}, \pi^{\pi'})$ (note that since π' is discrete, it also defines a permutation on the vertices of G). I.e., we identitfy the node ν with the invariant $(G^{\pi'}, \pi^{\pi'})$. As mentioned above, this does not influence any of the results of this paper directly – it only serves to make the description sound in terms of the further discussion in McKay & Piperno (2014). We refer to McKay & Piperno (2014) for an in-depth discussion of invariants.

B Relational Pooling

Relational pooling (RP) Murphy et al. (2019) is more general than discussed in this paper. In its initial formulation $\frac{1}{|V|!}\sum_{\pi \in S_{|V|}} f(G^\pi, X^\pi)$ it is not tractable and does thus not fit into the comparison considered here. In the original paper, three methods to make RP tractable are discussed. The first uses canonization, which itself is generally not tractable, and thus is not considered. The third uses a fixed point tractable (FPT) formulation (essentially describing a less expressive version of the k-dimensional WL algorithm), which is generally not universal unless the FPT parameter is not fixed, and even then its costs are polynomial in the FPT parameter which is computationally too expensive for this comparison. Only the second discussed option, which is originally motivated by stochastically estimating the initial formulation, is both universal and computationally practical, which is why it is considered here. In particular, it is also the only version implemented and tested in the original paper.

C Edge Coloring

We discuss the differences when using edge colors since the MUTAG data set contains edge labels, which we consider in the experiments.

In order to handle edge colors in DEJAVU for color refinement and random IR walks, we made the following modifications. We encode edge colors as vertex colors: we subdivide each edge of the graph using a vertex and color that vertex according to the color of the edge. Then, we ensured that the cell selector never selects nodes that were inserted to subdivide an edge, i.e., the solver can still only individualize vertices that truly correspond to the original vertices of the graph.

We want to note that there is a more efficient albeit more involved way of resolving the issue Piperno (2018).

Table 2: The hyperparameter space for bayesian hyperparameter optimization. Penalty describes how the parameter influences the hyperparameter optimization.

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
Minimum	8	32	1e-6	1e-10	16	2	1	0.01	1
Maximum	256	512	1e-2	1e-0.3	128	10	5	1.0	64
Data-Type	Integer	Integer	Real	Real	Integer	Integer	Integer	Real	Integer
Penalty	1-(x/256)	x/512	None	None	x/128	x/10	None	None	x/64

D Color Encoding

We discuss precisely how we encoded colors for the data sets discussed in this work. First, notice that we need to encode labels for nodes and for edges. We used the same method to encode colors for both. Almost all node and edge features can be considered binary numbers due to how the data sets are encoded. The only exception to this rule is the PROTEINS data set which has one natural number followed by a binary number. Thus, we simply read the node and edge features as a binary number, where the first bit has the highest order, and the last bit has the lowest order. This ensures that node and edge features are encoded as different natural numbers if the original features were different. However, due to the size of natural numbers, that specifically the NCI1 dataset produces, as it has 37 node features, we also apply a modulo operation by 12345.

E Experiments

Network architecture. Each MLP in each GIN layer has three layers (input, hidden, output) that widen to a fixed number of features as soon as possible and remain there throughout all subsequent layers. The input and hidden layers of every MLP are followed by a batch-norm operation. For graph classification tasks, we use global mean pooling followed by a linear transform and dropout with $p = 0.5$. We use the node embeddings after each GIN layer in this way and sum over all of them for the final graph representation. As activation functions, we use only ReLU. For node-classification tasks, we do the same as before without the global mean pooling.

Bayesian hyperparameter optimization. We optimize over the batch size, the number of epochs, the learning rate, weight decay, the number of features the GIN layers expand to and operate on, the number of GIN layers, the number of dimensions added for the node initializations, the fraction of epochs after which the learning rate gets decreased by 0.5 consecutively, and the number of random samples for EoR. Table 2 describes the hyperparameter space used for the experiments. For in indepth description of each parameter:

- Batch size refers to the batch size used during training. The smaller it is, the more it penalizes the evaluation metric during the bayesian hyperparameter search.
- Epochs refers to the number of epochs used during training. The bigger they are, the more they penalize.
- Learning rate refers to the initial learning rate used inside of the Adam optimizer.
- Weight decay refers to the typical weight decay parameter inside of the Adam optimizer.
- Features refer to the number of features the MLPs inside of the GIN layers expand to. Each node will have features many features after the first GIN layers first MLP layer. The more features, the more they penalize.
- Layers refer to the number of layers of the GIN networks. The more layers, the more they penalize.
- Dimensions refer to the number of dimensions each node's features are expanded by for the new features introduced by the different URF methods. This parameter is the same as the d in d-IRNI.

- Step size is a relative parameter that influences how the learning rate is changed during training depending on the number of epochs. For instance, if the number of epochs is set at 100, then a step size of 0.1 will mean the learning rate is divided by 0.5 every 10 epochs, a step size of 0.5 would mean the learning rate is divided by 0.5 once after 50 epochs, and a step size of 1 indicates the learning rate is never dropped.
- EoR refers to the number of random samples that are used to estimate the output of the network. For instance, for an EoR of 10 the input is modified 10 times i.i.d using the URF of choice. All 10 inputs are then passed through the network and the final predictions are then averaged for all 10 outputs.

Tables 4, . . . , 19 show the mean and standard deviation of the best-found hyperparameters across their seeds for all the datasets and methods. For the learning rate and weight decay, only the exponent is given.

Monte Carlo cross-validation. Each dataset is split using stratified 10-fold cross-validation with random shuffling. The first fold is used as the test set and the 9 others are used for bayesian hyperparameter optimization. After the best model is found, it is trained on all 9 folds and its performance on the test set is reported. This is repeated 10 times with different random shuffles. If the dataset initially already provides a test set, then the bayesian hyperparameter optimization is repeated for 10 different seeds. In the inner loop, which we referred to before as just bayesian hyperparameter optimization, the data is split using stratified 9-fold cross-validation. The first fold is used as a validation set and the other 8 folds are used to train the model, after which its performance is reported on the validation set. This is repeated 3 times with different random shuffles. This essentially estimates nested 10×9-fold cross-validation. The 10 and 3 were chosen based on a time budget. The performance on PROTEINS, MUTAG, and NCI1 is particularly sensitive to the variance in the performance estimate, so we expect to see an improved performance if more estimates are used.

Test system and time budget. The system that was used to do the experiments mentioned in this work is made up of:

- #60-Ubuntu SMP Tue Jul 2 18:22:20 UTC 2019 4.15.0-55-generic
- 2 Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz
- 754GiB System memory
- 10 GeForce RTX 2080 Ti

The experiments took approximately 20 days without testing and approximately 1 month with testing, where the machine was not used on full load always. Table 3 shows the computation time in seconds for 1 seed for each of the methods on each separate dataset.

Python libraries. We used python 3.8.10 to implement all the models and conduct all the experiments:

- dejavu-gi 0.1.3 (for IRNI(CR))
- networkx 2.6.3 (for constructing TRI and TRIX)
- numpy 1.21.4
- scikit-learn 1.0.1
- scikit-optimize 0.9.0 (for Bayesian hyperparameter optimization)
- scipy 1.7.3
- torch 1.10.0
- torch-geometric 2.0.2 (specifically for graph related machine learning)

Table 3: The time in seconds for the entire evaluation process of 1 seed of a GIN network with RNI, CLIP, RP, IRNI(CR), and without any of these on selected data sets. $_{EoR}$ indicates the use of ensembling over randomness.

Method	PROTEINS	MUTAG	NCI1	TRI	TRIX	EXP	CEXP	CSL
None	25672 ± 5426	7275 ± 5532	121487 ± 63237	10878 ± 2328	10809 ± 2288	20735 ± 9345	19652 ± 5739	2286 ± 409
RNI	34499 ± 16968	7802 ± 2471	101154 ± 29446	53897 ± 27607	46855 ± 21908	53381 ± 35114	58668 ± 19114	9999 ± 4512
CLIP	36189 ± 10374	12268 ± 3955	151603 ± 36904	52437 ± 19898	48608 ± 20161	65597 ± 13000	55558 ± 13902	10697 ± 3146
RP	26042 ± 8782	8783 ± 2234	126282 ± 36043	56640 ± 35035	48930 ± 34016	52605 ± 29282	53750 ± 25979	6321 ± 1917
IRNI	33383 ± 13685	7793 ± 1560	156347 ± 35831	42454 ± 13367	40597 ± 15643	64123 ± 13412	50042 ± 14155	10937 ± 1895
RNI_{EoR}	26593 ± 9390	7810 ± 3740	106401 ± 30762	26767 ± 10270	25630 ± 14628	34604 ± 12037	33747 ± 9654	8832 ± 6506
CLIP_{EoR}	65225 ± 31132	8418 ± 2048	118096 ± 17091	24487 ± 4718	28594 ± 6917	38565 ± 7954	82525 ± 17467	6810 ± 1807
RP_{EoR}	22203 ± 4547	6309 ± 2749	117571 ± 43888	21233 ± 5526	22385 ± 7073	31831 ± 13730	37978 ± 12934	6513 ± 2271
IRNI_{EoR}	33337 ± 12360	8204 ± 3483	160859 ± 55380	33799 ± 4711	61533 ± 21959	52990 ± 13094	75701 ± 18933	5528 ± 1360

Table 4: The mean and standard deviation for each found hyperparameter for each method on the PROTEINS dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	152.70 ± 83.17	278.10 ± 185.29	-3.21 ± 0.76	-5.12 ± 2.72	76.00 ± 35.85	5.80 ± 1.78	3.30 ± 1.68	0.53 ± 0.32
RNI	86.30 ± 82.32	303.50 ± 131.03	-3.53 ± 0.89	-5.72 ± 2.14	59.50 ± 41.13	5.30 ± 3.13	3.30 ± 1.27	0.63 ± 0.29
CLIP	141.90 ± 93.62	224.20 ± 140.10	-3.61 ± 1.06	-7.41 ± 2.70	60.40 ± 38.66	3.80 ± 1.33	3.60 ± 1.50	0.58 ± 0.30
RP	192.70 ± 52.98	262.50 ± 162.44	-2.90 ± 0.66	-6.76 ± 3.28	55.00 ± 35.34	2.80 ± 1.17	4.10 ± 1.04	0.55 ± 0.33
IRNI	141.70 ± 90.51	308.30 ± 165.17	-2.97 ± 1.17	-7.45 ± 2.51	72.40 ± 48.34	3.00 ± 1.34	3.20 ± 1.47	0.45 ± 0.25

Random Seed For the experiments we used the random seeds 0 through 9 as input to our code. However, our experiments might not be perfectly reproducible as DEJAVU the package we use to calculate random IR paths does not allow for its seed to be set.

F Trainability

Here we add an evaluation of the data from training all the models. More specifically, we consider the trainability of all the methods on the evaluated datasets. One aspect of trainability is: How easily can the model's hyperparameters be optimized? This question can be answered with BHO by evaluating how many hyperparameter optimization steps are necessary to reach a satisfactory performance. Alternatively, when comparing two models, the question can be answered by which model needs fewer steps to achieve greater performance. Considering our main experiment, we visualize the best-found performance at each BHO evaluation for each dataset and model. For each dataset and model, we have all BHO evaluations. From this, we compute each seed's best-found performance after each BHO step. We then average these seeds and visualize the BHO steps against the AUROC performance found (up to this point) in figures 4 and 5. We follow the nomenclature of table 1 in describing the datasets and the models. Note that the visualized performances are biased as we consider the maximum performance over multiple evaluations. Also, the best-found performances are not equal to the performance in table 1 as these figures use the validation set performances of the Monte Carlo cross-validation.

A clear difference is notable between the synthetic datasets TRI, TRIX, EXP, CEXP, and CSL and the practical datasets PROTEINS, MUTAG, and NCI1. Each model converges independently on the practical

Table 5: The mean and standard deviation for each found hyperparameter for each method on the MUTAG dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	157.10 ± 78.41	329.80 ± 152.31	-3.67 ± 0.63	-6.36 ± 2.25	72.30 ± 32.85	5.70 ± 3.07	2.90 ± 1.51	0.61 ± 0.30
RNI	145.10 ± 90.67	315.40 ± 150.22	-3.25 ± 0.78	-8.43 ± 2.22	55.40 ± 30.11	5.80 ± 2.79	3.60 ± 1.43	0.76 ± 0.24
CLIP	142.20 ± 94.73	208.90 ± 139.70	-3.11 ± 0.91	-7.12 ± 2.84	102.30 ± 17.73	6.30 ± 2.93	2.90 ± 1.14	0.68 ± 0.27
RP	181.90 ± 64.39	256.20 ± 131.57	-3.07 ± 0.77	-6.27 ± 3.09	55.10 ± 29.18	6.60 ± 2.97	3.80 ± 0.98	0.61 ± 0.30
IRNI	167.20 ± 78.88	371.80 ± 120.38	-3.21 ± 1.09	-5.81 ± 3.25	68.10 ± 40.89	5.80 ± 3.09	2.80 ± 1.47	0.62 ± 0.27

Table 6: The mean and standard deviation for each found hyperparameter for each method on the NCI1 dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	
None	75.20 ± 60.04	270.60 ± 138.40	-3.78 ± 0.85	-5.43 ± 2.02	91.10 ± 28.83	7.00 ± 2.05	3.40 ± 1.11	0.65 ± 0.32	1.00 ± 0.00
RNI	103.70 ± 59.22	309.80 ± 84.42	-2.97 ± 0.73	-5.22 ± 1.99	89.50 ± 32.23	6.00 ± 1.55	3.00 ± 1.73	0.62 ± 0.23	1.00 ± 0.00
CLIP	131.75 ± 58.16	403.25 ± 117.57	-3.28 ± 0.79	-5.79 ± 1.93	97.38 ± 30.59	7.38 ± 1.58	2.50 ± 1.66	0.50 ± 0.19	1.00 ± 0.00
ORNI	157.00 ± 81.81	362.70 ± 134.81	-3.15 ± 0.72	-6.06 ± 2.20	69.00 ± 22.95	5.40 ± 1.69	3.30 ± 1.19	0.45 ± 0.21	1.00 ± 0.00
IRNI	93.12 ± 83.31	369.38 ± 93.77	-3.88 ± 0.61	-6.40 ± 2.27	105.50 ± 26.80	8.12 ± 2.32	1.12 ± 0.33	0.57 ± 0.29	1.00 ± 0.00

Table 7: The mean and standard deviation for each found hyperparameter for each method on the TRI dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	256.00 ± 0.00	32.00 ± 0.00	-4.15 ± 1.90	-4.15 ± 4.71	16.00 ± 0.00	2.00 ± 0.00	3.80 ± 1.83	0.50 ± 0.49
RNI	132.50 ± 91.25	454.30 ± 55.92	-2.61 ± 0.48	-8.01 ± 2.22	92.10 ± 26.43	9.50 ± 0.67	1.60 ± 1.28	0.66 ± 0.29
CLIP	129.30 ± 105.04	436.60 ± 52.67	-2.68 ± 0.31	-7.31 ± 2.06	78.60 ± 25.15	8.40 ± 1.36	3.80 ± 0.75	0.54 ± 0.33
RP	161.40 ± 102.91	397.30 ± 98.27	-2.55 ± 0.38	-7.80 ± 1.65	80.60 ± 21.90	8.80 ± 1.40	4.40 ± 0.80	0.56 ± 0.22
IRNI	147.70 ± 93.12	437.30 ± 87.21	-2.67 ± 0.41	-6.83 ± 2.14	78.30 ± 27.22	8.10 ± 1.51	3.80 ± 0.98	0.68 ± 0.29

Table 8: The mean and standard deviation for each found hyperparameter for each method on the TRIX dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	256.00 ± 0.00	32.00 ± 0.00	-4.15 ± 1.90	-4.15 ± 4.71	16.00 ± 0.00	2.00 ± 0.00	3.80 ± 1.83	0.50 ± 0.49
RNI	126.40 ± 67.20	431.20 ± 43.66	-2.62 ± 0.30	-7.63 ± 2.11	94.80 ± 24.86	9.40 ± 0.80	2.00 ± 1.34	0.46 ± 0.19
CLIP	120.70 ± 83.98	378.60 ± 77.44	-2.65 ± 0.45	-7.15 ± 1.79	67.90 ± 16.12	8.10 ± 1.45	4.00 ± 0.89	0.64 ± 0.26
RP	168.20 ± 96.77	445.90 ± 54.74	-2.63 ± 0.44	-6.80 ± 1.63	80.40 ± 32.04	8.50 ± 1.12	4.40 ± 1.02	0.69 ± 0.28
IRNI	185.10 ± 83.23	451.80 ± 75.61	-2.59 ± 0.31	-9.16 ± 1.23	76.40 ± 29.88	8.90 ± 1.30	4.10 ± 0.94	0.63 ± 0.21

Table 9: The mean and standard deviation for each found hyperparameter for each method on the EXP dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	205.90 ± 67.09	172.10 ± 166.92	-4.29 ± 1.82	-4.40 ± 3.54	52.10 ± 40.55	4.30 ± 2.33	2.60 ± 1.80	0.56 ± 0.39
RNI	164.40 ± 84.80	425.10 ± 79.47	-2.95 ± 0.50	-7.35 ± 1.75	63.00 ± 37.26	8.80 ± 1.40	2.30 ± 1.68	0.64 ± 0.28
CLIP	223.20 ± 51.71	307.30 ± 120.73	-2.78 ± 0.59	-7.11 ± 2.50	63.50 ± 41.48	7.60 ± 1.20	1.80 ± 1.17	0.42 ± 0.23
RP	139.10 ± 86.74	378.10 ± 115.58	-3.59 ± 0.40	-7.03 ± 2.21	84.90 ± 41.42	8.80 ± 1.25	4.70 ± 0.46	0.52 ± 0.24
IRNI	174.50 ± 78.69	210.80 ± 159.69	-3.20 ± 0.38	-7.32 ± 2.63	33.20 ± 12.40	8.20 ± 1.54	1.80 ± 1.08	0.61 ± 0.25

Table 10: The mean and standard deviation for each found hyperparameter for each method on the CEXP dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	149.70 ± 86.61	143.60 ± 98.14	-2.51 ± 0.54	-5.26 ± 2.77	49.20 ± 30.00	6.80 ± 2.23	2.70 ± 1.42	0.33 ± 0.25
RNI	153.40 ± 86.97	430.10 ± 102.38	-3.69 ± 0.31	-6.46 ± 2.01	63.40 ± 36.21	9.10 ± 1.45	1.40 ± 0.92	0.87 ± 0.14
CLIP	190.50 ± 49.88	308.00 ± 90.24	-3.38 ± 0.77	-7.61 ± 2.04	54.30 ± 30.84	8.30 ± 1.55	2.10 ± 1.37	0.66 ± 0.23
RP	130.30 ± 85.78	353.30 ± 99.67	-3.65 ± 0.28	-5.25 ± 2.11	75.90 ± 27.93	8.80 ± 1.08	4.60 ± 0.66	0.53 ± 0.27
IRNI	169.90 ± 87.46	245.80 ± 157.52	-3.13 ± 0.89	-6.69 ± 1.85	49.00 ± 34.81	8.00 ± 2.05	2.80 ± 1.54	0.51 ± 0.30

Table 11: The mean and standard deviation for each found hyperparameter for each method on the CSL dataset without EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size
None	256.00 ± 0.00	32.00 ± 0.00	-4.15 ± 1.90	-4.15 ± 4.71	16.00 ± 0.00	2.00 ± 0.00	3.80 ± 1.83	0.50 ± 0.49
RNI	74.90 ± 86.57	481.70 ± 58.11	-3.15 ± 0.45	-5.27 ± 2.27	90.00 ± 35.89	8.30 ± 1.35	1.70 ± 1.42	0.62 ± 0.26
CLIP	179.10 ± 73.78	312.10 ± 91.59	-2.25 ± 0.35	-6.33 ± 2.00	66.40 ± 36.34	7.00 ± 1.67	3.50 ± 1.12	0.58 ± 0.18
RP	185.00 ± 70.37	319.30 ± 88.94	-2.50 ± 0.48	-6.31 ± 2.48	53.00 ± 29.85	7.50 ± 1.43	4.20 ± 0.87	0.42 ± 0.24
IRNI	191.90 ± 58.75	240.00 ± 114.25	-2.86 ± 0.53	-6.67 ± 2.83	37.70 ± 14.21	4.70 ± 1.62	4.00 ± 0.89	0.41 ± 0.23

Table 12: The mean and standard deviation for each found hyperparameter for each method on the PROTEINS dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	180.80 ± 72.66	265.90 ± 147.07	-3.98 ± 1.03	-6.88 ± 2.31	62.10 ± 33.05	5.60 ± 2.87	4.60 ± 0.49	0.67 ± 0.30	36.10 ± 17.31
CLIP	92.10 ± 69.64	194.30 ± 153.41	-3.42 ± 1.00	-3.85 ± 2.41	63.70 ± 37.81	5.90 ± 2.77	3.60 ± 1.20	0.45 ± 0.30	21.00 ± 17.40
RP	157.00 ± 78.16	281.10 ± 153.80	-2.50 ± 0.28	-4.69 ± 3.19	66.00 ± 39.44	2.80 ± 1.17	2.90 ± 1.04	0.41 ± 0.25	41.60 ± 17.45
IRNI	163.70 ± 74.50	296.30 ± 169.70	-2.83 ± 1.00	-7.44 ± 2.49	57.50 ± 46.20	3.90 ± 1.76	3.80 ± 1.25	0.43 ± 0.41	18.50 ± 13.34

Table 13: The mean and standard deviation for each found hyperparameter for each method on the MUTAG dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	110.70 ± 80.39	221.50 ± 132.31	-3.57 ± 0.69	-6.83 ± 2.79	78.20 ± 35.47	6.40 ± 2.69	2.90 ± 1.45	0.59 ± 0.24	34.40 ± 15.88
CLIP	111.70 ± 88.57	269.30 ± 114.02	-3.65 ± 0.61	-4.71 ± 3.12	80.30 ± 34.81	6.90 ± 2.21	2.80 ± 1.25	0.42 ± 0.36	14.50 ± 14.12
RP	153.10 ± 79.12	255.70 ± 173.48	-2.96 ± 0.89	-5.14 ± 2.90	84.20 ± 46.18	7.30 ± 2.49	2.70 ± 1.10	0.68 ± 0.29	35.80 ± 16.77
IRNI	158.40 ± 84.20	310.00 ± 129.19	-3.65 ± 1.01	-4.81 ± 2.33	76.00 ± 43.38	5.60 ± 3.14	2.10 ± 1.37	0.61 ± 0.18	39.30 ± 21.87

Table 14: The mean and standard deviation for each found hyperparameter for each method on the NCI1 dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	153.00 ± 86.19	266.60 ± 120.89	-4.46 ± 0.33	-5.09 ± 1.85	96.10 ± 22.78	8.20 ± 1.54	3.20 ± 1.17	0.56 ± 0.29	44.60 ± 17.35
CLIP	162.50 ± 76.87	383.90 ± 109.63	-4.04 ± 0.77	-7.20 ± 2.36	86.70 ± 35.18	7.90 ± 1.87	2.40 ± 1.20	0.41 ± 0.31	35.30 ± 14.49
RP	138.10 ± 85.82	246.50 ± 59.54	-4.54 ± 0.41	-5.02 ± 2.15	102.60 ± 26.59	7.00 ± 1.90	3.60 ± 0.80	0.55 ± 0.30	43.00 ± 10.61
IRNI	174.10 ± 104.05	356.70 ± 137.42	-3.68 ± 0.47	-7.05 ± 2.42	102.60 ± 18.67	7.50 ± 2.01	1.80 ± 0.98	0.61 ± 0.33	7.50 ± 12.67

Table 15: The mean and standard deviation for each found hyperparameter for each method on the TRI dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	200.40 ± 56.35	236.70 ± 85.80	-2.54 ± 0.52	-5.66 ± 2.51	51.90 ± 29.79	5.90 ± 1.87	3.30 ± 1.19	0.58 ± 0.22	37.90 ± 17.17
CLIP	205.40 ± 43.87	219.50 ± 118.76	-2.79 ± 0.48	-5.77 ± 2.64	38.90 ± 25.93	4.50 ± 1.63	4.00 ± 1.18	0.58 ± 0.28	25.90 ± 16.35
RP	214.00 ± 47.44	214.90 ± 137.52	-2.43 ± 0.45	-7.03 ± 2.55	31.30 ± 22.45	5.30 ± 2.28	4.40 ± 0.66	0.44 ± 0.27	25.30 ± 15.07
IRNI	198.30 ± 69.54	170.00 ± 139.94	-2.72 ± 0.73	-6.79 ± 2.77	30.70 ± 12.45	3.90 ± 1.45	3.60 ± 1.62	0.31 ± 0.29	34.40 ± 15.81

Table 16: The mean and standard deviation for each found hyperparameter for each method on the TRIX dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	203.30 ± 60.54	264.40 ± 93.60	-2.50 ± 0.44	-6.91 ± 2.52	45.20 ± 22.30	5.70 ± 2.33	3.20 ± 1.08	0.54 ± 0.24	31.60 ± 14.83
CLIP	187.10 ± 73.91	140.70 ± 97.46	-2.94 ± 0.54	-6.43 ± 3.27	49.60 ± 29.86	4.30 ± 1.35	3.80 ± 1.33	0.62 ± 0.32	36.50 ± 16.41
RP	190.20 ± 78.03	182.80 ± 124.46	-3.16 ± 0.68	-5.02 ± 2.93	53.10 ± 27.94	3.80 ± 1.54	4.00 ± 1.18	0.64 ± 0.18	32.20 ± 18.42
IRNI	199.20 ± 57.18	187.90 ± 111.60	-2.60 ± 0.62	-5.01 ± 2.37	52.30 ± 21.19	5.50 ± 2.25	4.20 ± 1.17	0.55 ± 0.27	26.10 ± 16.02

Table 17: The mean and standard deviation for each found hyperparameter for each method on the EXP dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	221.50 ± 53.90	359.50 ± 91.93	-3.35 ± 0.58	-6.44 ± 3.08	63.60 ± 35.13	7.70 ± 1.35	2.50 ± 1.28	0.68 ± 0.35	29.30 ± 20.25
CLIP	196.80 ± 53.20	225.50 ± 93.81	-2.94 ± 0.33	-6.43 ± 2.73	51.70 ± 20.40	4.80 ± 1.60	3.20 ± 1.54	0.49 ± 0.26	25.60 ± 9.79
RP	198.60 ± 72.36	249.90 ± 107.72	-3.30 ± 0.72	-6.20 ± 2.35	53.20 ± 29.50	6.90 ± 1.64	3.40 ± 0.92	0.60 ± 0.30	23.10 ± 16.59
IRNI	165.80 ± 70.00	236.20 ± 120.13	-3.28 ± 0.89	-6.77 ± 1.84	44.70 ± 19.11	7.20 ± 2.09	2.60 ± 1.11	0.48 ± 0.28	20.60 ± 19.02

Table 18: The mean and standard deviation for each found hyperparameter for each method on the CEXP dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	172.10 ± 66.95	404.80 ± 66.97	-3.78 ± 0.39	-6.29 ± 2.17	77.00 ± 36.91	8.30 ± 1.55	1.30 ± 0.46	0.74 ± 0.24	22.50 ± 19.99
CLIP	193.70 ± 64.15	304.90 ± 121.05	-3.05 ± 0.72	-5.41 ± 2.55	44.90 ± 24.97	7.90 ± 0.94	2.40 ± 1.02	0.70 ± 0.32	23.80 ± 19.20
RP	193.70 ± 79.71	235.20 ± 96.69	-3.57 ± 0.46	-6.56 ± 2.48	68.70 ± 40.74	7.60 ± 1.56	3.20 ± 1.33	0.73 ± 0.26	25.30 ± 13.50
IRNI	190.40 ± 63.72	224.10 ± 155.54	-3.02 ± 0.67	-5.86 ± 3.11	46.60 ± 21.89	8.10 ± 1.58	2.40 ± 1.20	0.53 ± 0.32	16.20 ± 14.04

Table 19: The mean and standard deviation for each found hyperparameter for each method on the CSL dataset with EoR

Parameter	batch size	epochs	learning rate	weight decay	features	layers	dimensions	step size	EoR
RNI	164.50 ± 105.35	464.60 ± 102.96	-3.18 ± 0.58	-7.46 ± 2.07	89.70 ± 41.53	9.10 ± 1.04	1.70 ± 1.42	0.58 ± 0.23	50.50 ± 10.13
CLIP	190.70 ± 73.12	366.50 ± 124.09	-2.55 ± 0.45	-7.67 ± 1.34	59.10 ± 29.38	6.70 ± 1.00	3.70 ± 1.49	0.37 ± 0.25	16.20 ± 14.53
RP	187.40 ± 75.99	319.70 ± 92.42	-2.39 ± 0.42	-6.95 ± 2.11	70.80 ± 23.32	7.20 ± 0.98	3.20 ± 1.40	0.54 ± 0.28	11.50 ± 11.60
IRNI	193.40 ± 54.69	217.40 ± 74.28	-2.63 ± 0.45	-6.22 ± 3.23	37.40 ± 14.84	6.20 ± 1.94	3.90 ± 1.04	0.43 ± 0.32	17.10 ± 11.57

datasets to its optimal performance, making comparisons between the models more difficult. On the synthetic datasets, this comparison is more straightforward, as the models mostly converge to the same optimum. Notably, RNI converges slower than all other models. Additionally, considering the other models, a rough order of trainability could be surmised to be: (1) IRNI, (2) CLIP, (3) ORNI, (4) RNI, however, this order is less significant than RNIs poor trainability. This order would coincide with the amount of randomness in each model. If we consider for each model the size of the IR tree on any given graph, then the order of size would be the same. IRNI requires the smallest amount of individualizations to reach completely distinguished graphs. ORNI and RNI have maximal trees since they ignore any color information present during the IR tree construction. CLIP's first step is the same as IRNI, and then it can be roughly compared to ORNI and RNI, so its tree size is in between the others. Lastly, RNI can be considered more random than ORNI since it uses continuous random variables, which results in a random space of infinite size. The possible graphs for IRNI, CLIP, and ORNI are always finite. From this, we conclude making use of less randomness improves trainability.

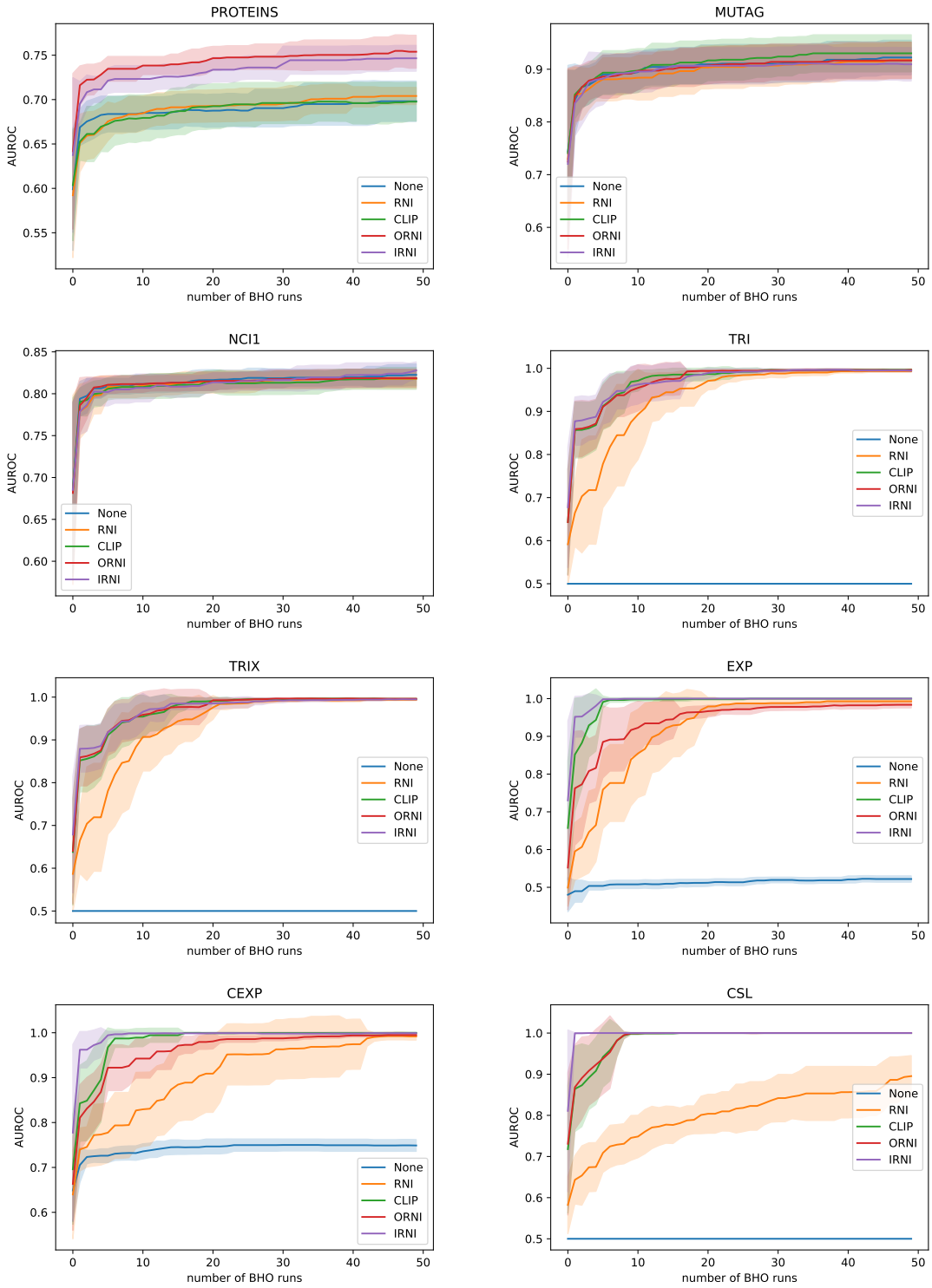

Figure 4: Plots showing the best performance (y) after x BHO steps (x). For each method 10 seeds are shows.

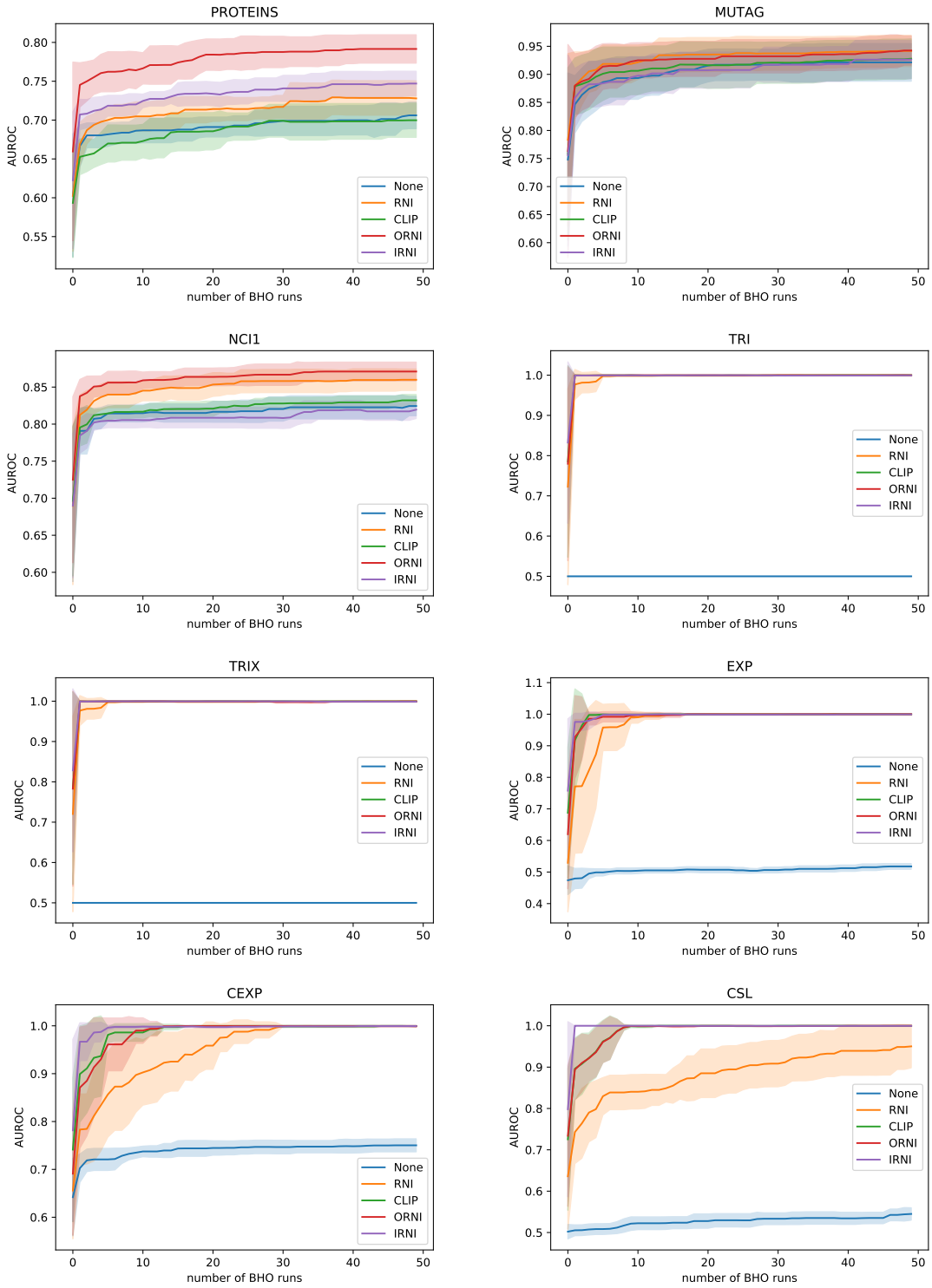

Figure 5: Plots showing the best performance (y) after x BHO steps (x). For each method 10 seeds are shows. Methods here use EoR.