A Characterization of Individualization-Refinement Trees

Markus Anders
TU Darmstadt, Germany

Jendrik Brachter
TU Darmstadt, Germany

Pascal Schweitzer
TU Darmstadt, Germany

Abstract

Individualization-Refinement (IR) algorithms form the standard method and currently the only practical method for symmetry computations of graphs and combinatorial objects in general. Through backtracking, on each graph an IR-algorithm implicitly creates an IR-tree whose order is the determining factor of the running time of the algorithm.

We give a precise and constructive characterization which trees are IR-trees. This characterization is applicable both when the tree is regarded as an uncolored object but also when regarded as a colored object where vertex colors stem from a node invariant. We also provide a construction that given a tree produces a corresponding graph whenever possible. This provides a constructive proof that our necessary conditions are also sufficient for the characterization.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Mathematics of computing → Trees

Keywords and phrases individualization refinement algorithms, backtracking trees, graph isomorphism

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.24

Related Version *Full Version*: `https://arxiv.org/abs/2109.07302` [2]

Funding Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant No. 820148).

1 Introduction

The individualization-refinement (IR) framework is a general backtracking technique employed by algorithms solving tasks related to the computation of symmetries of combinatorial objects [16]. These include algorithms computing automorphism groups, isomorphism solvers, canonical labeling tools used for computing normal forms, and to some extent recently also machine learning computations in convolutional neural networks [1,17]. In fact all competitive graph isomorphism/automorphism solvers, specifically NAUTY/TRACES [15,16], BLISS [11,12], SAUCY [8,9], CONAUTO [13,14], and DEJAVU [3,4] fall within the framework. These tools alternate color-refinement techniques (such as the 1-dimensional Weisfeiler-Leman algorithm) with backtracking steps. The latter perform artificial individualization of indistinguishable vertices. This leads to recursive branching and overall to a tree of recursive function calls, the so called IR-tree.

Using clever invariants and heuristics, the tools manage to prune large parts of the IR-tree. Since the non-recursive work is quasi-linear, it has long been known that the number of traversed nodes of the IR-tree is the determining factor in the running time for all the tools (see for example [20, Theorem 9] and [19]). And in fact, the running times of the various tools closely reflect this [4,16]. Indeed, variation in the traversal strategies among the tools leads to a different number of traversed nodes which in turn leads to different running times. However, explicit bounds that rigorously show asymptotic advantages of randomized traversals over deterministic ones have only recently been obtained [5]. For this, a specific problem – a search problem in trees with symmetries – is defined. It captures precisely the parameters within which IR-algorithms operate.

While these results are quite general within an abstract model, the bounds proven in [5] apply to the search problem in arbitrary trees with symmetries, independent of whether they originate from actual IR-computations or not. Granted, the vast benchmark library of Traces [15,16] shows that IR-trees come in an abundance of forms and shapes. However, to date there have been no comprehensive results actually analyzing which trees can arise as an IR-tree.

Contribution. In this paper we study which trees are IR-trees. Arising from a branching process, all IR-trees are rooted and all inner vertices have at least 2 children. Such trees are called irreducible (or series reduced). Despite a vast variety of IR-trees arising from benchmark libraries, it turns out that not all irreducible trees are IR-trees. However, we can give a full, constructive characterization of IR-trees.

▶ **Theorem 1.** *An irreducible tree is an IR-tree if and only if there is no node that has exactly two children of which exactly one is a leaf.*

To prove the theorem we first provide and justify necessary conditions for a tree to be an IR-tree. We then prove that, indeed, these conditions are sufficient by providing graphs on which the execution of an IR-algorithm yields the desired tree. In fact, our proof is constructive, meaning that we obtain an algorithm with the following property. Given a tree T satisfying the necessary conditions, the algorithm produces a graph whose IR-tree is T.

As we describe in our definition of IR-trees in Section 2, the trees are naturally associated with a coloring of the vertices. This coloring is a crucial component that is related to the automorphism group structure of the graph. Our characterization also fully describes how color classes may be distributed in a given tree. It turns out that there are several simple restrictions, in particular for vertices that have precisely two children, but apart from that all colorings can be realized and in particular any number of symmetries can be ensured (see Section 4).

Our characterization provides a fundamental argument transferring the analysis of abstract tree traversal strategies performed in [5] to backtracking trees of IR-algorithms on actual instances. Specifically, we may conclude that the abstract trees used for the lower bounds of probabilistic algorithms in [5] indeed appear as IR-tees. However, interestingly, the abstract trees used for the lower bounds of deterministic algorithms (Theorem 13, [5]) are not IR-trees. In fact these trees have nodes with two children, one child that is a leaf and another that is not. This breaks the necessary conditions as laid out by Theorem 1. Fortunately, it also immediately follows from our results that a slight modification can rectify this: by simply replacing the respective leaves with inner nodes that have two attached leaves, the trees become actual IR-trees, due to our characterization. Overall, we therefore prove that the lower bounds of [5] hold true in the IR-paradigm.

Cell Selectors and Invariants. Formally, the IR-paradigm allows for different design choices in some of its components. For most of these, competitive practical solvers actually make very similar choices: the refinement is always *color refinement* and solvers commonly choose as their pruning invariant (essentially) the so-called *quotient graph.* The way in which the actual implementations differ from color refinement and quotient graphs is usually only in minor details and done to achieve practical speed-ups. This only leads to a slightly weaker refinement and invariants in some specific cases. In this paper, we therefore comply with these common design choices.

Many other design choices, such as *how* IR-trees are traversed, have no effect on the characterization of the IR-trees themselves.

There is however one integral design choice where competitive IR-solvers do indeed vary in a way that affects which trees are IR-trees, namely the so-called *cell selectors.* We should emphasize that Theorem 1 only says that for the trees satisfying the necessary conditions there is some cell selector for which the graph is an IR-tree.

However, we can also say something about specific cell selectors. Considering the characterization for a *given* cell selector, there are two possibilities: either, fewer trees turn out to be IR-trees or the same characterization applies. We can use our results to argue that for some cell selectors that are used in practice our necessary conditions are sufficient, while for others they are not (see Section 5 for a discussion).

Techniques. Many properties of a graph, e.g. symmetries, are directly tied to properties of its IR-tree. When modeling a graph that is supposed to produce a particular IR-tree, two major difficulties arise, roughly summarized as follows:

1. The effect of color refinement on the graph needs to be kept under control.
2. The shape of the IR-tree may dictate that symmetries must be simultaneously represented in distinct parts of the graph.

We resolve these issues using various gadget constructions specifically crafted for this purpose. We introduce *concealed edges*, which allow us to precisely control the point in time at which the IR-process is able to see a certain set of edges and thus color refinement to take effect (resolving issue (1)). By combining concealed edges with gadgets enforcing particular regular abelian automorphism groups we can synchronize symmetries across multiple branches of the tree (resolving issue (2)).

Here, as the main tool we show the following. As an additional restriction, which stems from the structure of IR-trees, we consider only trees where all leaves can be mapped to the same number of other leaves via symmetries (i.e., under automorphisms all *leaf orbits* have the same size). We show that each such tree T can be embedded into a graph H_T, such that H_T restricts the symmetries of T in a particular way. Intuitively, we keep just enough symmetries to allow leaves to be mapped to each other whenever this is possible in T. We thereby effectively couple leaf orbits so that when fixing one leaf, all other leaves are fixed as well. More formally we prove the following theorem.

▶ **Theorem 2.** *Let T be a colored tree in which all leaf orbits have the same size. There exists a graph H_T containing T as an automorphism invariant induced subgraph so that the action of $\mathrm{Aut}(H_T)$ is faithful on T and semiregular on the set of leaves of T. Moreover, $\mathrm{Aut}(H_T)$ induces the same orbits on T as $\mathrm{Aut}(T)$.*

Again, we prove the theorem in a constructive manner. All steps can be easily converted into an algorithm that takes as input an admissible (i.e., compatible with our necessary conditions from Section 3) colored tree T and produces a graph and cell selector with IR-tree T.

2 Individualization-Refinement Trees

Following [16] closely, we introduce the notion of an IR-tree. Algorithms based on the IR-paradigm explore these trees using various traversal strategies to solve graph isomorphism, graph automorphism or canonical labeling problems.

Colored Graphs. An undirected, finite graph $G = (V, E)$ consists of a set of vertices $V \subseteq \mathbb{N}$ and a set of edges $E \subseteq V^2$, where E is symmetric. Set $n := |V|$.

The IR framework relies on coloring the vertices of a graph. A coloring is a surjective map $\pi\colon V \to \{1, \dots, k\}$. The i-th *cell* for $i \in \{1, \dots, k\}$ is $\pi^{-1}(i) \subseteq V$. Elements in the same cell are *indistinguishable.* If $|\pi| = n$, i.e., whenever each vertex has its own distinct color in π, then π is called *discrete.* A coloring π is *finer* than π' (and π' *coarser* than π) if $\pi(v) = \pi(v')$ implies $\pi'(v) = \pi'(v')$ for all $v, v' \in V$. Whenever convenient, we may also view colorings as ordered partitions instead of maps. A colored graph (G, π) consists of a graph and a coloring.

The symmetric group on $\{1, \dots, n\}$ is denoted $\mathrm{Sym}(n)$. An automorphism of a graph G is a bijective map $\varphi\colon V \to V$ with $G^\varphi := (\varphi(V), \varphi(E)) = (V, E) = G$. With $\mathrm{Aut}(G)$ we denote the automorphism group of G. For a colored graph (G, π) we require automorphisms to also preserve colors, i.e., $\pi(v) = \pi(\varphi(v))$ for all $v \in V$. We define the colored automorphism group $\mathrm{Aut}(G, \pi)$ accordingly.

Color Refinement and Individualization. IR-algorithms use a procedure to heuristically refine colorings based on the degree of vertices. The intuition is that if two vertices have different degree, then they can not be mapped to each other by an automorphism. We assign vertices of different degrees distinct colors to indicate this phenomenon. This process is iterated using color degrees: for example, two vertices can only be mapped to each other if they have the same number of neighbors of a particular color i. Therefore vertices can be distinguished according to the number of neighbors they have in color i. This gives us a new, refined coloring that (potentially) distinguishes more vertices. This is repeated until the process stabilizes.

The colorings resulting from this process are called equitable colorings. A coloring π is *equitable* if for every pair of (not necessarily distinct) colors $i, j \in \{1, \dots, k\}$ the number of j-colored neighbors is the same for all i-colored vertices. For a colored graph (G, π) there is (up to renaming of colors) a unique coarsest equitable coloring finer than π [16]. We denote this coloring by $\mathrm{Ref}(G, \pi, \epsilon)$, where ϵ is the empty sequence.

IR-algorithms also use *individualization.* This process artificially forces a vertex into its own cell. We can record which vertices have been individualized in a sequence $\nu \in V^*$. We extend the refinement function so that $\mathrm{Ref}(G, \pi, \nu)$ is the unique coarsest equitable coloring finer than π in which every vertex in ν is a singleton with its own artificial color. Specifically, the artificial colors used to individualize ν are not interchangeable with colors introduced by the refinement itself and are ordered: the i-th vertex in ν is always colored using the i-th artificial color.

We require this coloring to be isomorphism invariant (which means that $\mathrm{Ref}(G, \pi, \nu)(v) = \mathrm{Ref}(G^\varphi, \pi^\varphi, \nu^\varphi)(v^\varphi)$ for $\varphi \in \mathrm{Sym}(n)$). There are efficient *color refinement* algorithms to compute $\mathrm{Ref}(G, \pi, \nu)$, for which we refer to [16].

We say two colored graphs (G_1, π_1) and (G_2, π_2) are *distinguishable (by color refinement)*, if with respect to the colorings $\mathrm{Ref}(G_1, \pi_1, \epsilon)$ and $\mathrm{Ref}(G_2, \pi_2, \epsilon)$

1. there is a color c with differently sized cells in G_1 and G_2 (i.e., $|\text{Ref}(G_1, \pi_1, \epsilon)^{-1}(c)| \neq |\text{Ref}(G_2, \pi_2, \epsilon)^{-1}(c)|$),
2. or there are vertices $v_1 \in V(G_1)$, $v_2 \in V(G_2)$ of the same color (i.e., $\text{Ref}(G_1, \pi_1, \epsilon)(v_1) = \text{Ref}(G_2, \pi_2, \epsilon)(v_2)$), such that there is a color c within which v_1 and v_2 have a differing number of neighbors (i.e., $|\{(v_1, w) \in E(G_1) \mid \text{Ref}(G_1, \pi_1, \epsilon)(w) = c\}| \neq |\{(v_2, w) \in E(G_2) \mid \text{Ref}(G_2, \pi_2, \epsilon)(w) = c\}|$).

Sequences (or t-tuples) of vertices $\nu_1 \in (G_1, \pi_1)^t$ and $\nu_2 \in (G_2, \pi_2)^t$ are *distinguishable*, if the graphs $(G_1, \text{Ref}(G_1, \pi_1, \nu_1))$ and $(G_2, \text{Ref}(G_2, \pi_2, \nu_2))$ are.

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. After individualization, color refinement is applied again and the process continues recursively. Formally, a cell selector is a function $\text{Sel}\colon \mathcal{G} \times \Pi \to 2^V$ (where $\mathcal{G}$ denotes the set of all graphs and Π denotes the set of all colorings), satisfying:

- Isomorphism invariance, i.e., $\text{Sel}(G^\varphi, \pi^\varphi) = \text{Sel}(G, \pi)^\varphi$ for $\varphi \in \text{Sym}(n)$.
- If π is discrete then $\text{Sel}(G, \pi) = \emptyset$.
- If π is not discrete then $|\text{Sel}(G, \pi)| > 1$ and $\text{Sel}(G, \pi)$ is a cell of π.

IR-Tree. We describe the IR-tree $\Gamma_{\text{Sel}}(G, \pi)$ of a colored graph (G, π), which depends on a chosen cell selector Sel. Essentially, IR-Trees simply describe the call-trees stemming from the aforementioned backtracking procedure. Nodes of the search tree are sequences of vertices of G. The root of $\Gamma_{\text{Sel}}(G, \pi)$ is the empty sequence ϵ. If ν is a node in $\Gamma_{\text{Sel}}(G, \pi)$ and $C = \text{Sel}(G, \text{Ref}(G, \pi, \nu))$, then the set of children of ν is $\{\nu.v \mid v \in C\}$, i.e., all extensions of ν by one vertex v of C.

By $\Gamma_{\text{Sel}}(G, \pi, \nu)$ we denote the subtree of $\Gamma_{\text{Sel}}(G, \pi)$ rooted in ν. We omit the index Sel when apparent from context.

We recite the following fact on isomorphism invariance of the search tree as given in [16], which follows from the isomorphism invariance of Sel and Ref:

▶ **Lemma 3.** *If ν is a node of $\Gamma(G, \pi)$ and $\varphi \in \text{Aut}(G, \pi)$, then ν^φ is a node of $\Gamma(G, \pi)$ and $\Gamma(G, \pi, \nu)^\varphi = \Gamma(G, \pi, \nu^\varphi)$.*

Quotient Graph. The IR-tree itself can be exponentially large in the order of G [18]. To decrease its size IR-algorithms use a pruning mechanism. For this a *node invariant* is used. A node invariant is a function $\text{Inv}\colon \mathcal{G} \times \Pi \times V^* \to I$ that assigns to each sequence of nodes of the tree a value in a totally ordered set I. It satisfies the following.

- Isomorphism invariance, i.e., $\text{Inv}(G, \pi, \nu_1) = \text{Inv}(G^\varphi, \pi^\varphi, \nu_1^\varphi)$ for $\varphi \in \text{Sym}(n)$.
- If $|\nu_1| = |\nu_2|$ and $\text{Inv}(G, \pi, \nu_1) < \text{Inv}(G, \pi, \nu_2)$, then for all nodes $\nu_1' \in \Gamma(G, \pi, \nu_1)$ and $\nu_2' \in \Gamma(G, \pi, \nu_2)$ it holds that $\text{Inv}(G, \pi, \nu_1') < \text{Inv}(G, \pi, \nu_2')$.

The particular way the node invariant can be exploited depends on the problem to be solved. When solving for graph isomorphism, the algorithm may prune all nodes with an invariant differing from an arbitrary node invariant. However, when algorithms want to compute a canonical labeling, they must find a specific canonical node invariant to continue with. However, in the context of the present work these details are not important.

Most IR-algorithms use a specific invariant, the so-called *quotient graph*, which is naturally produced by color refinement.

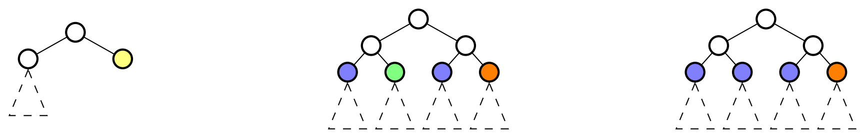


Figure 1 Forbidden structures in asymmetric binary IR-trees.

For an equitable coloring π of a graph G, the quotient graph $Q(G, \pi)$ captures the information of how many neighbors vertices from one cell have in another cell. Quotient graphs are complete directed graphs in which each vertex has a self-loop. They include vertex colors as well as edge colors. The vertex set of $Q(G, \pi)$ is the set of all colors of (G, π), i.e., $V(Q(G, \pi)) := \pi(V(G))$. The vertices are colored with the color of the cell they represent in G. We color the edge (c_1, c_2) with the number of neighbors a vertex of cell c_1 has in cell c_2 (possibly $c_1 = c_2$). Since π is equitable, all vertices of c_1 have the same number of neighbors in c_2.

A crucial fact is that graphs are *indistinguishable by color refinement* if and only if their quotient graphs on the coarsest equitable coloring are equal.

We should also remark that quotient graphs are indeed *complete invariants*, yielding the following property.

▶ **Lemma 4.** *Let ν, ν' be leaves of $\Gamma(G, \pi)$. There exists an automorphism $\varphi \in \mathrm{Aut}(G, \pi)$ with $\nu = \varphi(\nu')$ if and only if $Q(G, \mathrm{Ref}(G, \pi, \nu)) = Q(G, \mathrm{Ref}(G, \pi, \nu'))$.*

Consistent with the colors of trees used in [5], we may also view quotient graphs as a way to color IR-trees themselves, i.e., where we color a node ν with $Q(G, \mathrm{Ref}(G, \pi, \nu))$.

3 Necessary Conditions for IR-Trees

We collect necessary conditions for the structure of IR-trees. Since IR-trees are the result of a branching process, they are naturally irreducible (no node has exactly one child). Also, indistinguishable leaves can be mapped to each other.

▶ **Lemma 5.** *IR-trees are irreducible.*

▶ **Lemma 6.** *Let l_1, l_2 be two leaves of an IR-tree (T, π). If l_1 and l_2 are indistinguishable, there is an automorphism $\varphi \in \mathrm{Aut}(T, \pi)$ mapping l_1 to l_2.*

▶ **Lemma 7** (see e.g. [4])**.** *A leaf l can be mapped to exactly $|\mathrm{Aut}(G, \pi)|$ leaves in $\Gamma(G, \pi)$ using elements of the automorphism group $\mathrm{Aut}(G, \pi)$.*

It follows that all classes of indistinguishable leaves have equal size.

Since in color refinement, partitionings and hence quotient graphs only ever become finer and more expressive, the following properties hold.

▶ **Lemma 8.** *Let n_1, n_2 be two nodes of an IR-tree where n_i is on level l_i.*
1. *If $l_1 \neq l_2$, then n_1 and n_2 are distinguishable.*
2. *Consider the two walks starting in the root and ending in n_1 and in n_2, respectively. If in these walks two nodes on the same level are distinguishable then n_1 and n_2 are distinguishable.*

Some further restrictions apply specifically in the case of cells of size 2.

▶ **Lemma 9** (Forbidden Binary Structures).

1. *If a node n has two children n_1 and n_2, then it cannot be that exactly one of the children n_1 or n_2 is a leaf (see Figure 1, left).*
2. *If n_1, n_2 are any two nodes and n_1 has exactly 2 children then the multiset of colors of the children of n_1 and n_2 are equal or disjoint (Figure 1, middle and right).*

Proof. Part 1 follows from the fact that individualizing one vertex in a cell of size 2 also individualizes the other vertex of the cell.

For Part 2 we note that individualization of a child of n_1 also individualizes the other child of n_1 and vice versa. This implies that if a child c_2 of n_2 has the same color as some child c_1 of n_1, then by definition, individualization of c_1 and c_2, respectively, produces indistinguishable colorings. So in this case there is a one-to-one correspondence between the colors of the children of n_1 and those of n_2. ◀

It is easy to see that if at any point the cell selector chooses differently sized cells in different branches, the branches subsequently become distinguishable. However, if we assume cell selectors only base their decision on the quotient graph, this restriction applies earlier. More specifically, we call a cell selector *quotient-graph-based*, whenever the result of the cell selector depends only on the quotient graph rather than other aspects of G and π (i.e., we have $\mathrm{Sel}(Q(G, \pi))$ rather than $\mathrm{Sel}(G, \pi)$). Then, we have the following.

▶ **Lemma 10.** *If two nodes n and n' in an IR-tree are indistinguishable, then their parents have the same number of children. If additionally the cell selector is quotient-graph-based then n and n' also have the same number of children.*

Restricting the cell selector to quotient graphs thus changes whether we can distinguish nodes with a differing number of children *before* or *after* individualizing one more vertex. We may even distinguish cells *before* individualization in both cases, if we include the decision of the cell selector into the invariant itself (i.e., using $(Q(G, \pi), \mathrm{Sel}(G, \pi))$ instead of $Q(G, \pi)$, which is clearly only more expressive in case the cell selector is *not* quotient-graph-based).

In the following, we assume cell selectors are indeed quotient-graph-based. Since we only require a less powerful cell selector, our construction becomes more general. However, in the construction, we could alternatively drop the additional restriction above with minor adjustments by allowing a more powerful cell selector.

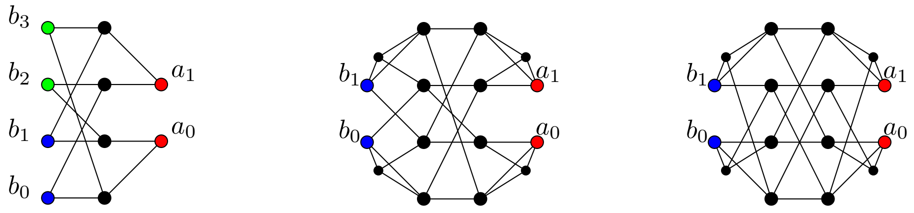
For the remainder of this paper we say that a tree fulfills the *necessary conditions*, if none of the conditions laid out by this section are violated.

4 Graph Constructions

Given a colored tree (T, π) which satisfies the necessary conditions, we describe how to construct a graph $G(T, \pi)$ and a cell selector $S(T, \pi)$ for which (T, π) is (up to renaming of colors) the IR-tree $\Gamma_{S(T,\pi)}(G(T, \pi))$. We make abundant use of gadget constructions, which we describe first. We give a concise description of the construction, details (i.e., the long version) can be found in the full version of the paper [2] (including omitted correctness arguments).

4.1 Gadgets

All our gadgets have multiple *input* and *output gates*. Each gate is a pair of vertices that together form their own color class in the gadget. Vertices in the gates are the only vertices of the gadgets connected to other vertices outside the gadget. We say that vertices labeled



(a) AND$_2$ gadget. **(b)** Unidirectional gadget. **(c)** Dead End gadget.

Figure 2 The AND$_2$ gadget and two variants of directional gadgets.

with b_i denote the "input", while a_i indicates "output". Gates can be *activated*, by which we mean the process of distinguishing the vertices of the gate pair into distinct color classes, and applying color refinement afterwards. We say activation *discretizes* the gadget if the resulting stable coloring on the gadget vertices is discrete.

Three of the gadgets we present (AND$_i$, Unidirectional and Dead End gadget) have already been used in other contexts related to color refinement [6, 7, 10].

AND$_i$ Gadget [6, 7, 10]. The AND$_2$ gadget as illustrated in Figure 2a, realizes the logical conjunction of gates with respect to color refinement, and an XOR gadget with respect to automorphisms.

Given $i > 2$, we can realize an AND$_i$ gadget with i input gates by combining multiple AND$_2$ gadgets in a tree-like fashion. The AND$_i$ gadget is constructed by attaching the first and second input gate to an AND$_2$, whose output is connected to another AND$_2$ together with the third input gate, and so on. We use colors to order the input gates, i.e., we color the i-th input gate with color i.

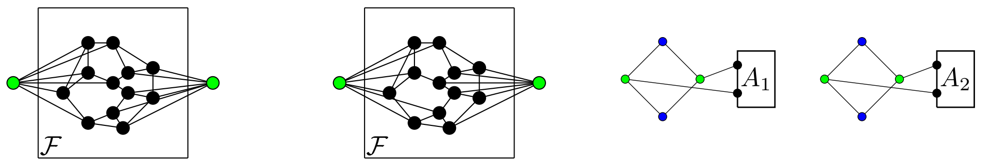
We define the special case of the AND$_1$ gadget to simply consist of a pair of vertices that functions as the input and output gate at the same time.

▶ **Lemma 11** ([10]). *The* AND$_i$ *gadget admits automorphisms that flip the output gate and either one of the input gates while fixing other input gates. As long as some input gate remains unsplit, the output gate is not split but activating all inputs discretizes the gadget .*

Unidirectional and Dead End Gadget [6, 10]. Next, we describe gadgets through which gate activation can be propagated or blocked depending on the direction of the gadget. Specifically we construct the unidirectional gadget (Figure 2b) and the dead end gadget (Figure 2c). Note that the two gadgets are indistinguishable from each other by color refinement. The smaller vertices depicted in Figure 2 have been included to guarantee that the gadgets become discrete after the input and output gate has been split and can otherwise be ignored.

▶ **Lemma 12.** *The unidirectional and dead end gadget are indistinguishable by color refinement. In the unidirectional case, activating the input discretizes the gate but activating the output does not split the input gate. In the dead end case, both input and output have to be activated to discretize the gadget.*

Asymmetry Gadgets. Our next gadgets only have one gate (see Figure 3, $\mathcal{F}$ denotes the Frucht graph). The crucial property of the asymmetry gadgets A_1 and A_2 (Figures 3a and 3b) is for the two gate vertices to be initially indistinguishable by color refinement, but to ensure that individualizing one of the gate vertices leads to a different quotient graph than individualizing the other gate vertex.



(a) Asym. gadget A_1. **(b)** Asym. gadget A_2. **(c)** A true edge. **(d)** A fake edge.

Figure 3 Asymmetry gadgets and concealed edges based on asymmetry gadgets.

▶ **Lemma 13.** *A_1 and A_2 are asymmetric and stable under color refinement. Activating the input gate discretizes the gadget and we obtain two non-isomorphic colorings depending on which vertex was individualized. Furthermore, $A_1 \ncong A_2$.*

Concealed Edges. Lastly, we describe the *concealed edge gadget* that is used to hide edges from color refinement. The gadget has two vertices that represent the endpoints of an edge (the blue vertices in Figure 3c). The idea is that instead of an edge connecting the two vertices, we insert a concealed edge gadget. For this the gadget has a pair consisting of two *inner vertices* (the green vertices in Figure 3c), which are both connected to each input vertex. This pair is then connected to an asymmetry gadget. We define two classes of edges, where one type of edge attaches the gadget A_1 (*true edges*) and the other A_2 (*fake edges*).

The crucial property is that as long as inner vertices of the gadgets are not distinguished, color refinement can not distinguish between true edges and fake edges. However, if we distinguish the inner vertices, true edges can indeed be distinguished from fake edges.

We always employ this gadget within the following design pattern. Whenever we want to connect two sets of vertices V_1 and V_2 with edges $E \subseteq V_1 \times V_2$ in a concealed manner, we first add a concealed edge gadget between *all* pairs $(v_1, v_2) \in V_1 \times V_2$. However, only if $(v_1, v_2) \in E$, we use a *true edge*, and whenever $(v_1, v_2) \notin E$ we use a *fake edge*. Finally, we connect all pairs of inner vertices of the concealed edge gadgets to some construction that is used to *reveal* the edges.

The asymmetry gadget prohibits automorphisms from flipping the concealed edge gadget itself. However, care has to be taken when connecting the inner vertices to other constructions: it is imperative to connect the inner vertices of multiple concealed edge gadgets that are on the, say, left side of the asymmetry gadget, in the same manner. Otherwise, once revealed, edges could possibly be distinguished into even more categories than just fake and true edges.

4.2 A construction for asymmetric trees

For our construction, we first restrict ourselves to asymmetric trees, i.e., all leaves have different colors. Building on this, the following section takes symmetries into account. Let (T, π) be an *asymmetric*, colored tree that satisfies the necessary conditions (see Section 3). We describe a graph $G(T, \pi)$ and cell selector $S(T, \pi)$ such that (T, π) is (up to renaming of colors) the IR-tree $\Gamma_{S(T,\pi)}(G(T, \pi))$.

The goal is to model the graph and cell selector in such a way that there is a one-to-one correspondence between paths in T and sequences of individualizations in $G(T, \pi)$. Such sequences are precisely the paths in the IR-tree $\Gamma_{S(T,\pi)}(G(T, \pi))$. To guarantee this correspondence, certain properties of paths in T must translate into specific properties for their respective sequence of individualizations. In particular, when modeling $G(T, \pi)$, we ensure the following.

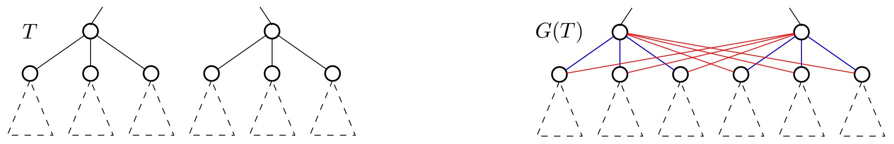


Figure 4 Connecting levels of the selector tree. Blue/red edges on the right symbolize true/fake edge gadgets.

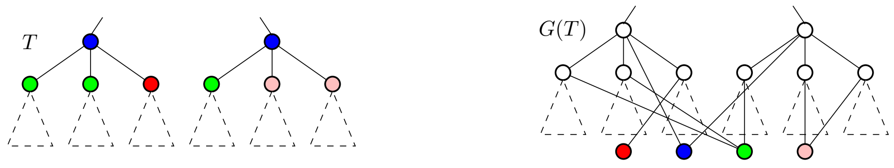


Figure 5 Colors of T are encoded in $G(T)$ by concealed edges to special *color nodes*.

1. Two paths must end in nodes of different color exactly if the corresponding sequences of individualizations result in different quotient graphs.
2. A path must end in a leaf exactly if the corresponding sequence of individualizations (when followed by color refinement) results in a discrete coloring.

Selector Tree. We start by describing the part of the graph on which the cell selector operates, i.e., within which cells are chosen. We use a copy of T itself for this purpose. One of the central difficulties is that color refinement executed on T may actually result in a coloring that is finer than π.

Hence, the selector tree consists of the nodes of T but encodes the edges of T in the selector tree using concealed edges (see Figure 4). This guarantees that our copy of T is stable under color refinement. At some point, we will need to add another gadget construction to ensure that edges between the levels are actually revealed at the right time. Assuming this for now, the cell selector $S(T, \pi)$ always chooses as next cell the cell that consists of the children of the node chosen last.

Colors. Next, we consider the colors π of T. Colors indicate whether a sequence of individualizations should lead to differing quotient graphs. We make use of fake edges again to encode this: we encode a one-to-one correspondence between selector tree nodes and their color in π using concealed edges (see Figure 5). We will discuss how edges are revealed further below. Specifically, we will reveal edges incident with node n at the point in time when node n is individualized.

Leaf Detection. Whenever we individualize a node n in the selector tree, we want to react to this by revealing a specific set of edges. Therefore, we now add a construction that detects whether a specific node n in a cell was individualized. Let $s \geq 2$ be the size of the current cell (in the tree T the current cell is always the set of children of some node). For each node n in the cell, consider all $s-1$ pairs with other nodes of the cell. We add an AND_{s-1} gadget and connect the left node of every input pair to n, and the other to one of the $s-1$ other nodes. An AND_{s-1} gadget is not symmetric in its input gates, so in order to keep things symmetrical, we add $(s-1)!$ many AND_{s-1} gadgets for every possible order of nodes in the input. We connect the output gates of the AND_{s-1} gadgets to the *individualization output* of n (see Figure 6). Crucially, the individualization output is activated (i.e., split) whenever n is individualized, and not immediately activated if only some other node n' is individualized.

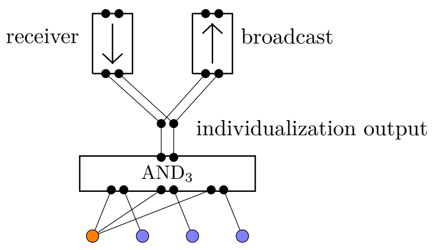


Figure 6 Leaf detection mechanism for the leftmost node of the cell. If this node is individualized, the AND$_3$ gadget is activated. The figure only shows true edges.

We again ensure the construction is stable under color refinement by using concealed edges. We use concealed edges between nodes of level i and the AND$_{s-1}$ of level i gadgets, using true edges instead of the edges described above.

If a node corresponds to a leaf and its individualization output is activated, we want to propagate discretization to other nodes. We add some control structures for every node n in the selector tree, namely a *broadcast* and *receiver* gadget as illustrated in Figure 6. We use a dead end gadget instead of a unidirectional as the broadcast gadget whenever n is not a leaf in T. Next, we connect the output of the broadcast gadget to the input of *all* receiver gadgets in the graph.

The idea is that if n is a leaf and individualized, we propagate this split to *all* individualization outputs through the broadcast and receiver gadgets, eventually causing a discretization of the graph, as explained below. If n is not a leaf, the broadcast gadget is a dead end gadget and no further split occurs.

Revealing Edges. We now describe when concealed edges are revealed. Assume we are individualizing a node at level i of the tree. At this point, we need to reveal the edges in the selector tree connecting level i to level $i+1$. We use the individualization outputs at level i to reveal edges of the selector tree to level $i+1$: we connect every individualization output through a unidirectional gadget with the internal nodes of the concealed edges between level i and level $i+1$.

In order to activate the individualization outputs, we also need to reveal edges from level $i+1$ nodes to their AND$_{s-1}$ gadgets. Hence, we do the same construction as above, connecting the unidirectional gadgets we added on level i to reveal these edges on level $i+1$. For the first level of the selector tree (children of the root) we do not use concealed edges, such that the level is initially revealed.

Finally, the same technique is also used to reveal colors. The individualization output of node n reveals the concealed edges between n and the color nodes. This reveals the color of n whenever we individualize n. Due to broadcasting, *all* colors are revealed whenever a leaf is activated, leading to a discrete coloring.

4.3 Generating symmetries

We expand our construction so that it can also handle colored trees (T, π) with prescribed symmetries. As such, the graph $G(T, \pi)$ can also be build from a tree (T, π) that is not necessarily asymmetric. In this case, sequences of individualizations along root-to-leaf paths still produce the desired tree (T, π) as a subtree of $\Gamma_{S(T,\pi)}(G(T, \pi))$. However, $G(T, \pi)$ is supposed to become discrete after the IR-process reaches a leaf, but at this point the selector tree is only split up to orbits prescribed by T.

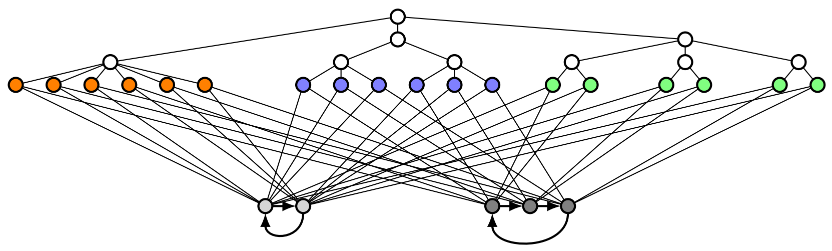


Figure 7 Symmetry cycles couple leaf orbits across multiple branches of the selector tree. The illustration omits fake edges. In the construction, cycles do not contain directed edges, but specially colored nodes that indicate direction.

Discretization of orbits is challenging since we need to make sure that the symmetries are not destroyed by the addition of new gadgets. Once leaf orbits have been discretized, discretization propagates through the selector tree and the whole construction becomes discrete.

Overall we need to construct the graph H_T mentioned in Theorem 2. To construct H_T, we introduce *symmetry cycles* and *symmetry couplings*. The basic idea is shown in Fig. 7, a detailed explanation can be found in the full version [2]. This in turn defines a new construction $\tilde{G}(T, \pi)$ by adding a concealed version of H_T to the selector tree.

Discretization of Orbits. We remark that the way we construct H_T has additional consequences on color refinement: individualization of a root-to-leaf path in H_T discretizes all symmetry cycles. In terms of $\tilde{G}(T, \pi)$, since different leaf orbits have different colors in T, this means that individualization of a root-to-leaf path now discretizes the set of leaves as well.

Concealing Symmetry Couplings. We hide H_T from color refinement using concealed edges. In the construction of H_T, we replace edges with true edge gadgets. All other connections between selector tree nodes and symmetry cycles become fake edges. To reveal the edges whenever a leaf is individualized, we connect the inner nodes of the concealed edges to the output of all broadcast gadgets.

5 Conclusion and Future Work

We have shown that every tree that meets some simple necessary conditions is an IR-tree. Regarding invariant pruning we should highlight that of course every pruned tree is a subtree of an unpruned tree, so our techniques extend to IR-algorithms with pruning.

Regarding refinement, we use the standard color refinement used by all IR-algorithms. However regarding cell selectors there is no clear standard. In this paper, we did not optimize the construction for any specific cell selector, but rather used the cell selector as part of the construction.

Let us now assume we are given a fixed cell selector. For a particular cell selector, there are two possibilities: either, fewer trees turn out to be IR-trees or the same necessary conditions apply. For the latter, we suspect that for many natural examples the construction of this paper can be adapted. Consider for example the cell selector that always chooses a smallest non-trivial cell. In this case, by adding more concealed structure enforcing specific cell sizes it can be shown that the same necessary conditions are indeed sufficient again.

In contrast to this, consider the cell selector that always chooses a largest non-trivial cell. Here, the degree of the vertices on root-to-leaf walks in a corresponding IR-tree must monotonically decrease. Hence, fewer trees turn out to be IR-trees and the necessary conditions are not sufficient. If interested in specific cell selectors one might therefore want to refine the necessary conditions.

Another interesting direction of research might be to investigate bounds for the order graphs realizing a given tree since this is related to the running time of IR-tools.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2112–2118. ijcai.org, 2021. doi:10.24963/ijcai.2021/291.

2 Markus Anders, Jendrik Brachter, and Pascal Schweitzer. A characterization of individualization-refinement trees. *CoRR*, abs/2109.07302, 2021. Full version of the paper. arXiv:2109.07302.

3 Markus Anders and Pascal Schweitzer. dejavu. URL: https://www.mathematik.tu-darmstadt.de/dejavu.

4 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 73–84. SIAM, 2021. doi:10.1137/1.9781611976472.6.

5 Markus Anders and Pascal Schweitzer. Search Problems in Trees with Symmetries: Near Optimal Traversal Strategies for Individualization-Refinement Algorithms. In *ICALP 2021*, volume 198 of *LIPIcs*, pages 16:1–16:21, 2021. doi:10.4230/LIPIcs.ICALP.2021.16.

6 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, Sebastian Kuhnert, and Gaurav Rattan. The parameterized complexity of fixing number and vertex individualization in graphs. In *MFCS2016*, volume 58 of *LIPIcs*, pages 13:1–13:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.13.

7 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.

8 Paul T. Darga, Hadi Katebi, Mark Liffiton, Igor L. Markov, and Karem Sakallah. Saucy3. URL: http://vlsicad.eecs.umich.edu/BK/SAUCY/.

9 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, Igor L. Markov, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 530–534, New York, NY, USA, 2004. ACM. doi:10.1145/996566.996712.

10 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. In *FOCS '96*, pages 264–273. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548485.

11 Tommi Junttila and Petteri Kaski. bliss. URL: http://www.tcs.hut.fi/Software/bliss/.

12 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.13.

13 José Luis López-Presa, Antonio Fernández Anta, and Luis N. Chiroque. conauto2. URL: https://sites.google.com/site/giconauto/.

14 José Luis López-Presa, Luis Núñez Chiroque, and Antonio Fernández Anta. Novel techniques for automorphism group computation. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933 of *LNCS*, pages 296–307. Springer, 2013. doi:10.1007/978-3-642-38527-8_27.

15 Brendan D. McKay and Adolfo Piperno. nauty and Traces. URL: `http://pallini.di.uniroma1.it`.

16 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, 2014. `doi:10.1016/j.jsc.2013.09.003`.

17 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. In *AAAI 2019*, pages 4602–4609, 2019. `doi:10.1609/aaai.v33i01.33014602`.

18 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *STOC 2018*, pages 138–150. ACM, 2018. `doi:10.1145/3188745.3188900`.

19 Adolfo Piperno. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR*, abs/0804.4881, 2008. `arXiv:0804.4881`.

20 Pascal Schweitzer. *Problems of unknown complexity: graph isomorphism and Ramsey theoretic numbers.* Phd. thesis, Universität des Saarlandes, Germany, 2009.