# Graph Automorphism partition and Canonization in Polynomial time

Christos I. Karatzas

*Independent Researcher*

**Abstract**

We show that the Graph Isomorphism *(GI)* problem can be solved in polynomial $O(n^3 + n^2 m \log(n+m) + m^2 n \log(n+m) + m^3 \log(n+m))$ time, on simple connected graphs with $n$ vertices and $m$ edges. As fundamental part of the proof, we introduce a novel method, named *Symmetry Classification*, which computes a canonical automorphism partition of a simple connected graph in polynomial $O(n^2 + nm \log n)$ time. The master algorithm reduces to bipartite graph isomorphism by transforming the input graph, if not bipartite, to its subdivision graph and computes the *canonical form* of the latter; or of the former if already bipartite. Canonical form is obtained by repeating the sequence of first applying *Symmetry Classification* method, then canonically selecting a vertex for *individualization*, and last applying the classical *1-dimensional Weisfeiler-Lehman* algorithm, until a canonical discrete coloring is attained. Since both bipartite and connected graph isomorphism are isomorphism complete we conclude that *GI* lies in *P*.

*Keywords:* Algorithms, Graph Canonization, Graph Isomorphism, Bipartite Graph Isomorphism, Graph Automorphism Partition, Polynomial Complexity

## 1. Introduction

### 1.1. Outline

This paper is organized as follows: In current section we introduce the basic notation, definitions and tools that will be used. In section 2 we present a novel method, which we name *Symmetry Classification*, that computes a canonical automorphism partition of a simple connected graph. In section 3 we present the *Master algorithm* along with proofs of correctness and analysis of its complexity. We conclude in section 4. In appendix *A* we include the listings of various subroutines that are being referenced in the main algorithms of this paper. In appendix *B* we showcase applications of the *Master algorithm* in selected graphs.

*Email address:* ckaratza@gmail.com (Christos I. Karatzas)

*1.2. Basic definitions and tools*

For the context of this paper let $G = (V, E)$ be a *simple* graph with set of vertices $V_G$ and set of edges $E_G$. By *simple* we mean undirected with no loops or multiple edges. We assume a normalized labeling of vertices $\{0, ..., |V_G| - 1\}$. The vertices adjacent to a vertex $u \in V_G$ form its *open neighborhood* $N_G(u)$. Any subgraph induced by a subset of vertices $S \subset V_G$ is denoted as $G[S]$. We may omit the subscript $G$ in cases where context is clear. Throughout this paper whenever we encounter lists of integers we assume that a sorted list obeys *ascending ordering* and the *comparator* between a pair of lists $L = [x_1, x_2, ..., x_k]$, $L^* = [x^*_1, x^*_2, ..., x^*_l]$ with cardinality $k$, $l$ respectively is: $L < L^*$ when $k < l$, and $L \leq L^*$ if there is some $j \leq k$ such that $x_i = x^*_i$ for $i < j$ and $x_j < x^*_j$. The comparator between a pair of lists of [list of lists] is defined in a similar manner. All algorithm listings imply zero indexed lists. Two graphs $G, H$ are said to be *isomorphic*, $G \cong H$, if there is a bijection $f : V_G \to V_H$ such that if $\{u, w\} \in E_G \iff \{f(u), f(w)\} \in E_H$. In this context $f$ is an *isomorphism* from $G$ to $H$. For a graph $G$ any bijection $A : V_G \to V_G$ such that if $\{u, w\} \in E_G \iff \{A(u), A(w)\} \in E_G$ is called *automorphism*. Mapping each vertex to itself is the *trivial automorphism*. The Graph Isomorphism *(GI)* problem asks to determine whether two given simple graphs are isomorphic.

**Definition 1.1.** Given a graph $G = (V, E)$, a set $S \subseteq V_G : |S| \geq 2$ along with its elements, is/are *symmetric*, if for any two $u, v \in S$ there is an *automorphism* $f : V_G \to V_G : f(u) = v$. A vertex $w$ which is mapped to itself for every possible automorphism is called *asymmetric*.

**Definition 1.2.** Given a graph $G = (V, E)$, vertices $u, v$ are called *pseudo-similar* if $G[V \setminus u] \cong G[V \setminus v]$ but $u, v$ are not *symmetric*.

**Definition 1.3.** A partition $\pi$ of a set $V$ is a set $\{V_1, ..., V_k\}$ of pairwise disjoint non-empty subsets of $V$, such as $\cup^k_{i=1} V_i = V$. The sets $V_i$ are called *blocks* of $\pi$. A partition $\pi$ is *discrete* if $|\pi| = |V|$ and *unit* if $|\pi| = 1$. Each block of size 1 is called *trivial*. Given a partition $\pi$, we write $u \sim_\pi v \iff \exists B \in \pi : u, v \in B$.

**Definition 1.4.** A partition $\kappa$ of $V$ refines a partition $\pi$ of $V$, if $\forall u, v \in V$ : $u \sim_\kappa v \Rightarrow u \sim_\pi v$.

**Definition 1.5.** Given a graph $G = (V, E)$, a partition $\pi$ of $V_G$ is *stable* if $\forall B \in \pi, \forall u, v \in V_G : u \sim_\pi v \Rightarrow |N(u) \cap B| = |N(v) \cap B|$. This means that any two vertices of a non-trivial block have the same number of neighbors that reside in any block.

**Proposition 1** ([1], Proposition 3)**.** Let $G = (V, E)$ be a graph. For every partition $\pi$ of $V_G$, there is a unique coarsest stable partition $\tau$ that refines $\pi$.

**Definition 1.6.** Let $G = (V, E)$ be a graph. A surjection $c : V_G \to \{1..k\}, k \leq n$ is called *k-coloring*. Given a *k-coloring* $c$ of $G$, we write $C^c_i = \{u \in V_G | c(u) = i\}$ and call it *color class i*. Likewise we denote the partition $\{C^c_i, ..., C^c_k\}$ by $\pi_c$. A *k-coloring* is characterized respectively from the attributes i.e., *discrete*,*unit*,*stable*, of its induced partition and vice versa. For brevity we may interchange naming k-coloring with just coloring in the course of this paper.

**Definition 1.7.** Given two graphs $G$, $H$ with respecting *k-colorings* $c_G$, $c_H$, an isomorphism $f$ from $G$ to $H$ is called *color preserving* for $c_G$, $c_H$, if $\forall u \in V_G \Rightarrow c_G(u) = c_H(f(u))$.

**Definition 1.8.** A *coloring method* is a method for obtaining a coloring $c_b$ of a graph $G$, given an initial coloring $c_a$.

**Definition 1.9.** A coloring method is called *canonical*, if for any two isomorphic graphs $G$ and $H$ with initial colorings $c_a^G$ resp. $c_a^H$ and isomorphism $f : V_G \to V_H$, the following holds: if $f$ is color preserving for $c_a^G$ and $c_a^H$, then $f$ is color preserving for the resulting colorings $c_b^G$ and $c_b^H$. Resulting colorings are also called *canonical colorings* of $G, H$ starting from $c_a^G$ resp. $c_a^H$. If starting coloring is the unit one, we simply call the resulting coloring a *canonical coloring* of a graph.

**Definition 1.10.** Given any two graphs $G$,$H$ with initial *k-colorings* $c_G$ resp. $c_H$ and an isomorphism $f$ from $G$ to $H$, we call *Canonical Color Refinement* (to be abbreviated as *CCR*) any canonical coloring method which implements *1-dimensional Weisfeiler-Leman algorithm* [2], that can compute the coarsest stable partitions $\pi_{c_G^*}$, $\pi_{c_H^*}$ which refine $\pi_{c_G}$, $\pi_{c_H}$. Both resulting colorings $c_G^*$, $c_H^*$ are unique 1 canonical l-colorings with $k \leq l$ starting from $c_G$ resp. $c_H$.

**Lemma 1.1** ([1], Theorem 9)**.** For any graph $G = (V, E)$ on $n$ vertices and $m$ edges given a *k-coloring* $c$ of G, there exists a *Canonical Color Refinement* algorithm that halts in $O((n + m) \log n)$ time.

**Definition 1.11.** Given a graph $G = (V, E)$ and a canonical stable coloring $c_G$ of $G$, we call *distinguished* all vertices in trivial blocks.

**Definition 1.12.** Given a graph $G = (V, E)$ and a *stable non-discrete* partition $\pi = \{B_1, ..., B_l\}$ of $G$, we call *individualization* of any vertex $u \in B_i : |B_i| > 1$, the process of transforming $\pi$ into $\pi^u = \{B_1, ..., B_i \setminus \{u\}, ...B_l, \{u\}\}$, or equivalently introducing a new color class $q = l + 1$ for $\{u\}$. By application of *CCR* on this *q-coloring*, we end up with a stable *k-coloring* $c_k$ where $k \geq q$. We call $c_k$ the *individualized coloring of* $u$ *on* $c_l$ and denote it as $Ind(u, c_l)$.

**Definition 1.13.** Given a simple graph $G = (V, E)$ on $n$ vertices, $m$ edges and a canonical discrete coloring $c_d$ of $G$, let $G(c_d)$ be an isomorphic copy of $G$, which is computed by relabeling the vertices of $G$ according to their order in $c_d$. We call $G(c_d)$ the *canonical form* of $G$, denoted as $Canon(G)$. The list of sorted open neighborhood label lists of each vertex of $Canon(G)$ starting from 0 until $n - 1$, is the *certificate* of $G$ denoted as $Cert(G)$. In case of vertices with zero degree, open neighborhood lists are empty. The routine of computing $Cert(G)$ is described in 6.

**Corollary 1.1.1.** Given two simple graphs $G, H$, if their certificates match then they are isomorphic.

*Proof.* By 1.13, since their certificates match their canonical forms are the same, thus they are also isomorphic. □

**Definition 1.14.** A *bipartite* graph is a graph whose vertices can be divided into two disjoint and independent sets $U_1$ and $U_2$, that is, every edge connects a vertex in $U_1$ to one in $U_2$. The routine which checks whether a simple connected graph is bipartite is described in 4.

**Definition 1.15.** An *edge subdivision* is the insertion of a new vertex $v_z$ in the middle of an existing edge $e = v_x v_y$ accompanied by the joining of the original edge endpoints with the new vertex, thus replacing edge $e$ with two edges $e_1 = v_x v_z$ and $e_2 = v_z v_y$.

**Definition 1.16.** Given a simple graph $G = (V, E)$ on $n$ vertices, $m$ edges, we call the *subdivision graph* $S(G)$ of $G$, the graph obtained by sequentially applying *edge subdivisions* 1.15 to all edges of $G$. Apparently $S(G)$ contains $n + m$ vertices and $2m$ edges. The routine of computing $S(G)$ is described in 5.

**Corollary 1.1.2.** Given a simple graph $G = (V, E)$ on $n$ vertices, $m$ edges, its subdivision graph $S(G)$ is a bipartite graph.

*Proof.* Follows from 1.14,1.16 , since for $S(G)$ the newly inserted $m$ vertices and existing $n$ vertices from $G$ form two disjoint independent sets, and all newly derived edges share their endpoints between these two sets. □

**Definition 1.17.** Given a simple connected graph $G = (V, E)$, a surjective coloring $c$ of $G$ and a vertex $u \in V$, we call *schema* $SC(u, c)$ of vertex $u$ on $c$, the output of the routine described in 3. The output is a list of sorted color lists which represent all levels of a Breadth First Traversal of graph $G$ (each list is one level), with vertex $u$ as root (level 1). Each level contains the colors of its vertices stemming from coloring $c$.

**Definition 1.18.** Given a simple connected graph $G = (V, E)$ and a surjective coloring $c$ of $G$, let $G[SC(u, c, l, l + 1)]$ be the subgraph induced by the vertices of levels $l, l+1$ of the schema $SC(u, c)$ of a vertex $u \in V$. We implicitly reference the vertices of each level since schema is a Breadth First Traversal routine, but doesn't contain vertices in its structure 3.

**Definition 1.19.** Given two simple graphs $G, H$, their *union* will be a graph $U$ with set of vertices $V_G \cup V_H$ and set of edges $E_G \cup E_H$.

## 2. Classification of symmetries

In this section we introduce a novel algorithm called *Symmetry Classification* (to be abbreviated $SYM$), which discovers the automorphism partition of a given simple connected graph.

### *2.1. Symmetry Classification Algorithm*

A canonical stable coloring $c_{start}$ of a simple connected graph $G = (V, E)$ and a surjection from each vertex $u$ of $G$ to a ranked integer value of the certificate of its open neighborhood's subgraph $Cert(G[N(u)])$, are given as input.

Ranking is based on a given comparator. Equal ranked values imply isomorphism. We assume that certificates have been generated by the same algorithm so that any comparison is relevant. Symmetry Classification 1 consists of two chained partition constructions and a final call to *Canonical Color Refinement* subroutine:

1. Construct a partition $\pi_{seed} = \{B_1, ..., B_l\}$ such that for each vertex $u, v$ $u \sim_{\pi_{seed}} v \iff c_{start}(u) = c_{start}(v) \land G[N(u)] \cong G[N(v)]$. Blocks of $\pi_{seed}$ are ordered, based first on their color value and then on their certificate ranking (numerical ordering). The induced coloring $c_{seed}$ is named *seed coloring*.
2. Construct a partition $\pi_{schema} = \{B_1, ..., B_k\}$ such that for each vertex $u, v$ $u \sim_{\pi_{schema}} v \iff SC(u, c_{seed}) = SC(u, c_{seed})$. Blocks of $\pi_{schema}$ are ordered, based on their schema value (list of [list of lists] comparator). The induced coloring $c_{schema}$ is named *schema coloring*.
3. Invoke *CCR* subroutine with $c_{schema}$ as input and obtain a stable coloring named *symmetry coloring*.

**Algorithm 1** Symmetry Classification (SYM)

**Input**:A simple connected Graph $G$ on $n$ vertices, $m$ edges, a canonical stable coloring $c_{\mathcal{S}}$ of $G$ and a surjection from each vertex $u$ of $G$ to a ranked value of the certificate of its open neighborhood's subgraph $Cert(G[N(u)])$.
**Output**: A canonical stable coloring of $G$ (symmetry coloring), which represents the automorphism partition of $G$.

```
function SYM($G$, $c_{\mathcal{S}}$, $CertMap$)
    $schemaMultiMap \leftarrow MultiMap\{\}$                                ▷ The schema multimap
    $coloringMap \leftarrow Map\{\}$                                        ▷ The color map of vertices
    $color \leftarrow 0$
    for $vertices$ in $c_{\mathcal{S}}.colorClasses$ do                     ▷ Classify based on color from $c_{\mathcal{S}}$
        if $vertices.size == 1$ then                                        ▷ Trivial block
            $coloringMap(vertices[0]) \leftarrow color$                     ▷ Trivial block inherited from input coloring
            $color \leftarrow color + 1$
            continue
        end if
        $vertices.sortWith(v_1, v_2 \rightarrow CertMap[v_1] \geq CertMap[v_2])$   ▷ Sort by certificate ranked value
        $coloringMap[vertices[0]] \leftarrow color$
        for $i$ in 1 until $vertices.length$ do
            if $CertMap[vertices[i]] > CertMap[vertices[i-1]]$ then
                $color \leftarrow color + 1$                                ▷ Different subgraph detected, new color assigned
                $coloringMap[vertices[i]] \leftarrow color$
            else
                $coloringMap[vertices[i]] \leftarrow color$
            end if
        end for
        $color \leftarrow color + 1$
    end for                                                                 ▷ Seed coloring calculated
    for $vertex$ in $G.vertices$ do                                         ▷ Compute Schema 3 of vertex based on seed coloring
        $schemaOfVertex \leftarrow SCHEMA(G, coloringMap, vertex)$
        $schemaMultiMap[schemaOfVertex] \leftarrow vertex$                  ▷ Add entry to multimap
    end for
    $sortedSchemata \leftarrow sort(schemaMultiMap.keys)$                   ▷ Sort schemata from multimap keys
    $coloringMap.clear()$                                                   ▷ Reset coloring map for schema partition
    $color \leftarrow 0$                                                    ▷ Reset color pointer
    for $schema$ in $sortedSchemata$ do                                     ▷ Construct schema partition based on schema ordering
        $vertices \leftarrow schemaMap[schema]$                             ▷ Get all vertices with this schema value
        for $vertex$ in $vertices$ do
            $coloringMap[vertex] \leftarrow color$
        end for
        $color \leftarrow color + 1$
    end for                                                                 ▷ Schema coloring calculated
    return $CCR(G, coloringMap)$                                            ▷ Apply CCR on schema coloring to obtain symmetry coloring
end function
```

**Lemma 2.1.** Given a simple connected graph $G = (V, E)$ and an initial canonical stable coloring $c$ of $G$, the symmetry coloring $c_s$ induced by application of

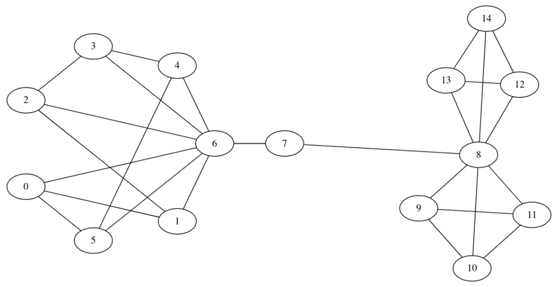
Symmetry Classification 1 on $c$ is stable and canonical.

*Proof. Seed coloring* 1 is canonical, because it (potentially) refines an initial canonical coloring and every isomorphism maps vertices to vertices of the same open neighborhood subgraph. Likewise *schema coloring* 1 is also canonical because its is derived from seed coloring and by definition of *Schema* 3, as a Breadth First Traversal method, every isomorphism maps vertices to vertices of the same schema. Since *schema coloring* is canonical, application of *Canonical Color Refinement* on it, yields also a canonical stable coloring 1.10, which in this context is the *symmetry coloring.* ☐

**Proposition 2.** Given a simple connected graph $G = (V, E)$ and an initial canonical stable coloring $c$ of $G$, the symmetry coloring $c_s$ induced by application of Symmetry Classification method 1 on $c$, classifies vertices as such:

1. By the isomorphism of their open neighborhood subgraphs.
2. By the number of neighbors that have the same open neighborhood subgraph.
3. By their *schema* value.
4. By the number of neighbors that have the same schema value.

*Proof.* The construction of *seed coloring* is a refinement of canonical stable coloring $c$, by further classifying vertices within classes of $c$ which have the same open neighborhood subgraph. The construction of *schema coloring* classifies vertices based on their schema value on *seed coloring* so conditions (1) and(3) of the corollary are met. Subsequent invocation of *Canonical Color Refinement* with *schema coloring* as starting coloring, leads to a stable coloring 1.10, which from 1.5 satisfies all four the conditions of the corollary. ☐



```
Symmetry coloring of graph → [[0, 1, 2, 3, 4, 5], [9, 10, 11, 12, 13, 14], [8], [6], [7]]
Schema of [0, 1, 2, 3, 4, 5] → [[1], [1, 1, 4], [1, 1, 1, 5], [3], [2, 2, 2, 2, 2, 2]]
Schema of [9, 10, 11, 12, 13, 14] → [[2], [2, 2, 3], [2, 2, 2, 5], [4], [1, 1, 1, 1, 1, 1]]
Schema of [8] → [[3], [2, 2, 2, 2, 2, 2, 5], [4], [1, 1, 1, 1, 1, 1]]
Schema of [6] → [[4], [1, 1, 1, 1, 1, 1, 5], [3], [2, 2, 2, 2, 2, 2]]
Schema of [7] → [[5], [3, 4], [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2]]
```

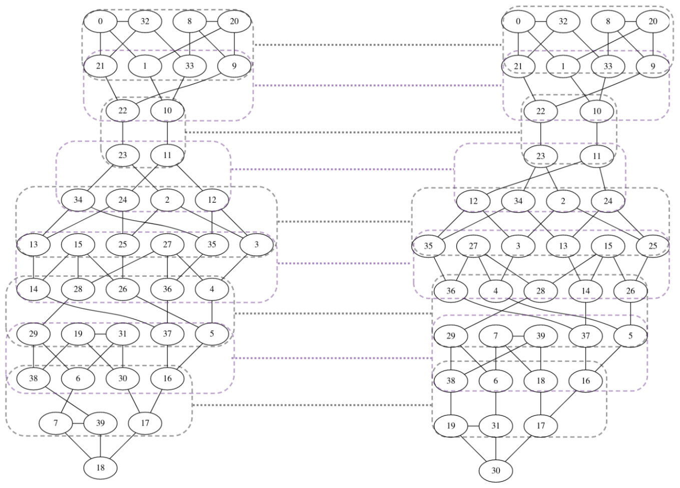
Figure 1: Schemata of all vertices on symmetry coloring

**Lemma 2.2.** Given a simple connected graph $G = (V, E)$ and a symmetry coloring $c_s$, it holds that $G[SC(u, c_s, l, l+1)]] \cong G[SC(v, c_s, l, l+1)]], \forall l \in \{1, .., |SC(v, c_s)| - 1\}, \forall u, v$ in the same block of $c_s$.

*Proof.* Let $u, v$ any vertices of the same block of $c_s$. For level 1 (root) and level 2 of $SC(u, c_s)$ and $SC(v, c_s)$, from 2 we have $G[SC(u, c_s, 1, 2)] \cong G[SC(v, c_s, 1, 2)]$ (1), because level 2 is the open neighborhood of $u$ and $v$. If level count of schemata is 2 then by (1) lemma holds. Let $N(t, l_1, l_2)$ denote the open neighborhood of vertex $t$ in levels $l_1, l_2$. For any vertices $a, b$ of level 2 of same color $c$ across both $SC(u, c_s), SC(v, c_s)$, by 2 we have $G[N(a)] \cong G[N(b)]$ (2). Let's assume that level count $\geq 3$ and $G[SC(u, c_s, 2, 3)] \ncong G[SC(v, c_s, 2, 3)]$ (3). Condition (1) implies $G[N(a, 1, 2)] \cong G[N(b, 1, 2)]$ (4), because both $a, b$ are connected to the same set of colors in level 2, thus in level 3 also. Since levels 1 and 3 have no adjacent vertices the subgraphs $G[N(a, 2, 3)], G[N(b, 2, 3)]$ because of (4) must be isomorphic in order to satisfy (2). But because of (3) there must exist some vertices $z, x$ of same color in level 2 of $SC(u, c_s), SC(v, c_s)$ respectively that $G[N(z, 2, 3)] \ncong G[N(x, 2, 3)]$ which is contradicting. Same for all remaining pairs of levels $\{(3, 4), ..(|SC(r, c_s)| - 1, |SC(r, c_s)|)\}, r \in u, v$ of both schemata. □

**Theorem 2.3.** *Given a simple connected graph $G = (V, E)$ and an initial canonical stable coloring $c$ of $G$, the symmetry coloring $c_s$ induced by application of Symmetry Classification method 1 on $c$ is the automorphism partition of $G$.*

*Proof.* By 2.2 for any two vertices $u, v$ of the same block $B_x$ we have that $G[SC(u, c_s, 1, 2)] \cong G[SC(v, c_s, 1, 2)]$ (1). If levels are 2 then by (1) the family of subgraphs $\{G[V \setminus u]\}_{u \in B_x}$ are all isomorphic. Let's assume that level count $\geq 3$ and that for the union 1.19 of levels $(1, 2)$ and $(2, 3)$ it holds that $\bigcup_{l=1}^{2} G[SC(u, c_s, l, l+1)] \ncong \bigcup_{l=1}^{2} G[SC(v, c_s, l, l+1)]$ (2). Condition (2) implies that for some vertices $a, b$ of the same color $c$ in level 2 it is $G[N(a)] \ncong G[N(b)]$ which is contradicting because of 2. If we continue taking the union of all subsequent levels we end up again with the family of subgraphs $\{G[V \setminus u]\}_{u \in B_x}$ being all isomorphic. The last step is to prove that vertices of the same block are not pseudo-similar 1.2 which is a direct consequence from the fact that they have the same schema 2. □



```
Symmetry coloring of graph → [[0, 7, 8, 19, 20, 31, 32, 39], [9, 18, 21, 30], [1, 6, 33, 38], [10, 29],
      [12, 15, 24, 27], [3, 4, 13, 14, 25, 26, 35, 36], [11, 28], [17, 22], [16, 23], [2, 5, 34, 37]]
```

Figure 2: Schemata (Breadth First Traversal) of symmetrical vertices 18 and 30 (second block of symmetry coloring) of the same graph depicting the isomorphism of their neighboring levels induced subgraphs. By taking the union of all pairs of consecutive levels induced subgraphs we end up with 2 isomorphic graphs $G[V \setminus 18], G[V \setminus 30]$.

**Lemma 2.4.** Given a simple connected graph $G = (V, E)$ on $n$ vertices, $m$ edges, Symmetry Classification Algorithm 1 takes $O(n^2 + nm \log n)$ time.

*Proof.* Sorting all vertices by the ranked value of the certificate of their open neighborhood subgraphs requires $O(n \log n)$ time. Constructing the seed coloring requires $O(n)$ time. Computing schemata of all vertices requires $O(n(n+m))$ time. Populating the multimap with schemata for all vertices as keys requires $O(nm)$ time, due to $O(m)$ time for hash computation. Sorting all schemata can take up to $O(nm \log n)$ since each schema has $O(m)$ size. Retrieving all vertices by their schema from multimap will require $O(nm)$. Building the schema coloring takes $O(n)$. Finally executing *Canonical Color Refinement* requires $O((n+m) \log n)$ 1.1. So overall complexity is $O(n \log n)+O(n)+O(n)+O(n(n+m))+O(nm \log n)+O(nm)+O(nm)+O((n+m) \log n) = O(n^2+nm \log n)$. $\square$

## 3. The Master Algorithm

The *Master Algorithm* 2 accepts a simple connected graph $G = (V, E)$ on $n$ vertices and $m$ edges as input, and calculates the certificate of the subdivision graph $S(G)$ 1.1.2, or of $G$ if it is already bipartite. The reduction to bipartite graph isomorphism is justified by the fact that ranking of certificates of open neighborhood subgraphs in a bipartite graph is computed in $O(m)$ time, since every subgraph consists of disconnected vertices with zero edges. Therefore their ranked value is equal to the degree of each vertex. The construction of seed coloring in 1 for the case of reduction to bipartite could be omitted, because the degree invariant is already covered by *CCR*. We choose to include it for clarity, because it doesn't affect the complexity of the algorithm. Since bipartite graph isomorphism is isomorphism complete [3] the only penalty involved is an increase in the number of vertices and edges up to $n + m$ and $2m$, in case $G$ is not already bipartite. The *Master Algorithm* starts by applying a *CCR* on the bipartite graph in order to obtain a canonical stable coloring $c_k$. If $c_k$ is discrete the algorithm returns the certificate of the reduced (or not) graph and halts. In any other case it iteratively applies the sequence *Symmetry Classification-Individualization-CCR* until a discrete coloring is attained. The choice of which vertex to individualize is performed by selecting any vertex from the minimum ordered color class of the canonical automorphism partition derived by *Symmetry Classification* that either isn't trivial or that is trivial but not already trivial in the starting (individualized) coloring of each round.

**Algorithm 2** Master Algorithm (MA)

**Input**:A simple connected graph $G$ on $n$ vertices, $m$ edges.
**Output**: $Cert(S(G))$ or $Cert(G)$, if $G$ is bipartite.

```
function MA($G$)
    $g \leftarrow G$                                      ▷ The graph to process. Initially set to $G$
    if !$ISBIPARTITE(G, 0)$ then                                      ▷ Check if $G$ is bipartite 4
        $g \leftarrow SUBDIVISION(G)$                          ▷ Compute subdivision graph $S(G)$ 5
    end if
    $certRanking \leftarrow Map\{\}$     ▷ vertices to ranked values of Open Neighborhood Subgraph surjection
    for $vertex$ in $g$ do
        $certRanking[vertex] \leftarrow vertex.degree$     ▷ Bipartite graph open neighborhood subgraphs are trivial
    end for
    $c_x \leftarrow CCR(g, c_{unit})$                                      ▷ Run CCR with unit coloring
    while true do
        if $c_x$ is discrete then                          ▷ If coloring is discrete break from loop
            break
        end if
        $autPartition \leftarrow SYM(g, c_x, certRanking)$ ▷ Calculate automorphism partition on current coloring 1
        $v_{ind} \leftarrow -1$                                  ▷ Vertex for individualization variable
        for $colorClass$ in $autPartition.colorClasses$ do ▷ Avoid individualizing already trivial vertex of $c_x$
            if !$colorClass.isTrivial$
                || $(colorClass.isTrivial$ && !$c_x.colorClasses[c_x(colorClass[0])].isTrivial)$ then
                $v_{ind} \leftarrow colorClass[0]$                    ▷ Select first vertex from color class
                $break$                                                   ▷ break from loop
            end if
        end for
        $c_x \leftarrow Ind(v_{ind}, c_x)$            ▷ Compute individualized coloring of selected vertex $v_{ind}$
    end while
    return $CERTIFICATE(g, c_x)$                                          ▷ Return certificate 6
end function
```

**Theorem 3.1.** *Given a simple connected graph $G = (V, E)$, the output obtained by applying Master Algorithm 2 on G is an invariant to test isomorphism.*

*Proof.* Since bipartite graph isomorphism is isomorphism complete [3], an invariant for subdivision graph $S(G)$ 1.1.2 is also valid for $G$. If *Canonical Color Refinement*, when initially invoked, induces a discrete coloring, then this is canonical 1.10. In any other case for each round we select any vertex from the minimum ordered class of symmetries that either isn't trivial (1) or is trivial but not already trivial in the current coloring (2). *Symmetry Classification* generates a canonical automorphism partition $\pi_{aut}$ 2.1 2.3 which implies that both (1) and (2) will be deterministically derived, because $\pi_{aut}$ will order all distinguished vertices also in a canonical way. Invoking *Symmetry Classification* in each round on an individualized coloring, even if its definition implies a canonical coloring of $G$ starting from unit 1, produces the automorphism partition of a graph $G_{ind}$ which is derived from $G$, by artificially making the individualized vertex $u_{ind}$ distinct (e.g. connecting a complete graph on $n$ vertices to $u_{ind}$), thus its output is valid for our canonicity context. Since in all cases we obtain a canonical discrete coloring, from 1.1.1 we conclude that Master Algorithm generates a valid invariant to test isomorphism. □

**Theorem 3.2.** *Given a simple connected graph $G = (V, E)$ on $n$ vertices, $m$ edges, Master Algorithm 2 takes $O(n^3 + n^2 m \log(n + m) + m^2 n \log(n + m) + m^3 \log(n + m))$ time.*

*Proof.* Checking if graph is bipartite requires $O(n + m)$ time. Computing subdivision graph $S(G)$ requires $O(n+3m)$ time. Since subdivision graph $S(G)$ has $n+m$ vertices and $2m$ edges we consider that all calculations are performed on $S(G)$. Calculating ranking of open neighborhood graphs of all vertices requires at most $O(n + 3m)$. Invoking *CCR* requires at most $O((n + 3m)\log(n + m))$ 1.1. *Symmetry Classification*, individualization selection and *CCR* (individualized colorings) can be invoked at most $n + m - 1$ times which takes $O((n + m - 1)((n + m)^2 + (n + m)2m \log(n + m))$ 2.4 $+$ $O((n + m - 1)(n + m))$ $+$ $O((n + m - 1)(n + 3m)\log(n + m))$. Computing the certificate of discrete coloring takes $O(n + 3m)$. So in total we have $O(n + m) + O(n + 3m) + O(n + 3m) + O(n + 3m) + O((n + m - 1)(n + m)) + O((n + m)(n + 3m)\log(n + m)) + O((n + m - 1)((n + m)^2 + (n + m)2m \log(n + m)) = O(n^3 + n^2 m \log(n + m) + m^2 n \log(n + m) + m^3 \log(n + m))$. □

**Theorem 3.3.** *Graph Isomorphism (GI) problem lies in P.*

*Proof.* Follows from 3.1, 3.2 and because connected graph isomorphism is isomorphism complete [3]. □

## 4. Conclusion

We have presented a new algorithm that computes an invariant to test graph isomorphism in polynomial time. The main new contribution was the introduction of *Symmetry Classification* algorithm which discovers the automorphism partition of the graph. Future work may include deeper analysis of the complexity, regarding various classes of graphs and comparison with other practical isomorphism algorithms [4]. In the same direction it is worth examining whether the structural results of this paper could provide a path for improving efficiency of Babai's Quasipolynomial Algorithm [5].

# Appendices

## A. Listings of subroutines used by the main algorithms

**Algorithm 3** Compute Schema of a vertex on a surjective coloring

**Input**:A simple connected graph $G$, a surjective coloring $c_x$ of $G$ and a vertex $u$ from a block of $c_x$.
**Output**: A list of sorted lists representing a Breadth First Traversal of $G$ with $u$ as root. Each level list contains the colors of its vertices stemming from $c_x$.

```
function SCHEMA($G$, $c_x$, $u$)
    $schema \leftarrow List\{\}$                          ▷ A jagged list containing the schema of the color class of $u$
    $visitedVertices \leftarrow Set\{\}$                  ▷ Keep track of visited vertices
    $bftQueue \leftarrow Queue\{\}$                       ▷ Queue to perform the Breadth First Traversal
    $visitedVertices.add(u)$                              ▷ Initialize structures with $u$
    $bftQueue.add()u)$
    while $bftQueue.isNotEmpty$ do
        $levelSize \leftarrow bftQueue.size$              ▷ Size of the current level
        $levelColorList \leftarrow List\{\}$              ▷ List that saves the current level
        while $levelSize \neq 0$ do
            $vertex \leftarrow bftQueue.pop()$
            $levelColorList.add(c_x(vertex))$             ▷ Map vertex to its color from $c_x$
            for $neighbor$ in $neighborhood(G, vertex)$ do
                if $!visitedVertices.contains(neighbor)$ then
                    $bftQueue.add(neighbor)$
                    $visitedVertices.add(neighbor)$
                end if
            end for
            $levelSize \leftarrow levelSize - 1$
        end while
        $schema.add(sort(levelColorList))$                ▷ Sort color list and add new level
    end while
    return $schema$
end function
```

**Algorithm 4** Check if graph is bipartite

**Input**:A simple connected graph $G$ on $n$ vertices, $m$ edges and the root vertex to start.
**Output**: A decision whether $G$ is bipartite.

```
function ISBIPARTITE($G$, $root$)
    $edgeCount \leftarrow G.edges.length$
    $vertexCount \leftarrow G.vertices.length$
    if $4 * edgeCount > (\frac{vertexCount}{2})^2$ then   ▷ Check edge, vertex condition if $G$ is bipartite
        return $false$
    end if
    $colorMap \leftarrow Map\{\}$                         ▷ Color map of 0-1 values
    $bftQueue \leftarrow Queue\{\}$                       ▷ Queue to perform the Breadth First Traversal
    $bftQueue.add(root)$                                  ▷ Initialize structures
    $colorMap[root] \leftarrow 0$
    while $bftQueue.isNotEmpty$ do
        $vertex \leftarrow bftQueue.pop()$
        $color \leftarrow colorMap[vertex]$
        for $neighbor$ in $neighborhood(G, vertex)$ do
            if $color == colorMap[neighbor]$ then         ▷ If colors match graph is not bipartite
                return $false$
            else
                $colorMap[neighbor] \leftarrow |color - 1|$  ▷ Assign different color
                $bftQueue.add(neighbor)$
            end if
        end for
    end while
    return $true$
end function
```

**Algorithm 5** Compute Subdivision graph

**Input**:A simple connected graph $G$ on $n$ vertices, $m$ edges and the root vertex to start.
**Output**: The subdivision graph $S(G)$.

```
function SUBDIVISION($G$)
    $subdivision \leftarrow SimpleGraph\{\}$                         ▷ The empty subdivision graph
    $visitedEdges \leftarrow Set\{\}$                              ▷ Keep track of visited edges of $G$
    for $vertex$ in $G.vertices$ do
        if $!subdivision.containsVertex(vertex)$ then
            $subdivision.addVertex(vertex)$
        end if
        for $neighbor$ in $neighborhood(G, vertex)$ do
            if $!subdivision.containsVertex(neighbor)$ then
                $subdivision.addVertex(neighbor)$
            end if
            if $!visited.contains(Edge(vertex, neighbor))$ then
                $subdivisionvertex \leftarrow Vertex\{\}$         ▷ New vertex to be added in subdivision graph
                $subdivision.addVertex(subdivisionVertex)$        ▷ Add new vertex
                $subdivision.addEdge(Edge(vertex, subdivisionVertex))$   ▷ Create two new edges
                $subdivision.addEdge(Edge(subdivisionVertex, neighbor))$
                $visited.add(Edge(vertex, neighbor))$             ▷ Mark edge of $G$ as visited
            end if
        end for
    end for
    return $subdivision$                 ▷ Return subdivision graph on $n + m$ vertices, $2m$ edges
end function
```

**Algorithm 6** Compute Certificate of Graph

**Input**:A simple graph $G$ on $n$ vertices, $m$ edges and a canonical discrete coloring $c_d$.
**Output**: The certificate $Cert(G)$.

```
function CERTIFICATE($G$, $c_d$)
    $gCopy \leftarrow SimpleGraph\{\}$                               ▷ The empty copy of graph $G$
    for $vertex$ in $G.vertices$ do
        $gCopy.addVertex(c_d(vertex))$                   ▷ Add relabeled vertices based on $c_d$
    end for
    for $vertex$ in $G.vertices$ do
        for $neighbor$ in $neighborhood(G, vertex)$ do
            $gCopy.addEdge(c_d(vertex), c_d(neighbor))$    ▷ Add relabeled edge endpoints based on $c_d$
        end for
    end for
    $certificate \leftarrow List\{\}$                   ▷ The list of sorted open neighborhood lists
    for $vertex$ in $gCopy.vertices$ do               ▷ Collection starts from vertex 0 until $n - 1$
        $neighbors \leftarrow sort(neighborhood(gCopy, vertex))$  ▷ sorted open neighborhood labels list of vertex
        $certificate.add(neighbors)$                              ▷ Add list to certificate
    end for
    return $certificate$                                      ▷ Return certificate of $G$
end function
```

## B. Application of Master Algorithm in selected Graphs

Below we showcase the application of *Master Algorithm* for some example graphs. The comparators used for sorting operations adhere to what is described in 1.2. For brevity we omit including the computation of subdivision graphs and the ranking of certificates for open neighborhood subgraphs. We use abbreviations for *Canonical Color Refinement (CCR)* and *Symmetry Classification (SYM)* subroutines. Canonical Color Refinement subroutine is implemented as defined in [1].

### *B.1. Miyazaki Graphs*

Miyazaki [6] graphs are regular and form special cases of Cai–Fürer–Immerman construction based on Fürer gadgets[7].

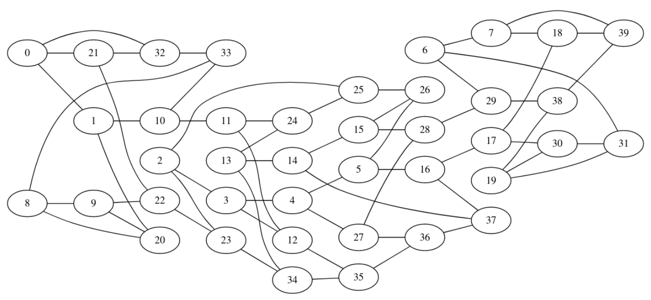


Figure 3: Miyazaki $M_1$ graph G

Listing 1: Application of Master Algorithm on Miyazaki graph $G$

```
[0]. Subdivision of input graph computed.
[0]. Ranking of open neighborhood subgraphs computed.

[1]. CCR→[[40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
      64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
      89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
      16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]]
[1]. SYM→[[42, 58, 60, 82], [41, 57, 59, 62, 80, 81, 85, 94], [40, 44, 54, 56, 61, 83, 95, 99], [63, 78,
      79, 84], [43, 55, 65, 93], [49, 51, 68, 70, 72, 74, 88, 91], [64, 92], [48, 69, 89, 97], [66, 67, 75,
       90], [0, 7, 8, 19, 20, 31, 32, 39], [9, 18, 21, 30], [1, 6, 33, 38], [10, 29], [12, 15, 24, 27], [3,
       4, 13, 14, 25, 26, 35, 36], [11, 28], [17, 22], [76, 86], [16, 23], [46, 52, 77, 87], [2, 5, 34,
      37], [45, 47, 50, 53, 71, 73, 96, 98]]
[1]. Individualize 42 for next round.

[2]. CCR→[[40, 95], [0, 32], [42], [19, 7, 39, 31], [82, 58], [1, 33], [21], [85, 41], [84], [22], [86],
      [23], [87, 46], [2, 34], [96, 71, 45, 47], [60], [20, 8], [59, 62], [44, 61], [9], [63], [3, 35, 25,
      13], [48, 97, 69, 89], [4, 36, 26, 14], [24, 12], [66, 67], [49, 68, 70, 88], [11], [64], [10], [65,
      43], [50, 98, 51, 53, 72, 73, 74, 91], [5, 37, 27, 15], [52, 90, 75, 77], [16, 28], [76, 92], [17,
      29], [55, 93, 78, 79], [18, 6, 38, 30], [80, 81, 83, 99, 54, 56, 57, 94]]
[2]. SYM→[[40, 95], [0, 32], [42], [7, 19, 31, 39], [58, 82], [1, 33], [21], [41, 85], [84], [22], [86],
      [23], [46, 87], [2, 34], [45, 47, 71, 96], [60], [8, 20], [59, 62], [44, 61], [9], [63], [3, 13, 25,
      35], [48, 69, 89, 97], [4, 14, 26, 36], [12, 24], [66, 67], [49, 68, 70, 88], [11], [64], [10], [43,
      65], [51, 72, 74, 91], [50, 53, 73, 98], [15, 27], [5, 37], [75, 90], [52, 77], [16], [76], [17],
      [78, 79], [55, 93], [18, 30], [6, 38], [57, 80, 81, 94], [54, 56, 83, 99], [28], [92], [29]]
[2]. Individualize 40 for next round.

[3]. CCR→[[95], [0], [42], [19, 7, 39, 31], [82, 58], [1], [21], [41], [84], [22], [86], [23], [87, 46],
      [2, 34], [96, 71, 45, 47], [60], [20], [62], [44], [9], [63], [3, 35, 25, 13], [48, 97, 69, 89], [4,
      36, 26, 14], [24, 12], [66, 67], [49, 68, 70, 88], [11], [64], [10], [43], [50, 98, 51, 53, 72, 73,
      74, 91], [5, 37, 27, 15], [52, 90, 75, 77], [16, 28], [76, 92], [17, 29], [55, 93, 78, 79], [18, 6,
      38, 30], [80, 81, 83, 99, 54, 56, 57, 94], [40], [32], [33], [61], [65], [8], [59], [85]]
[3]. SYM→[[95], [0], [42], [7, 19, 31, 39], [58, 82], [1], [21], [41], [84], [22], [86], [23], [46, 87],
      [2, 34], [45, 47, 71, 96], [60], [20], [62], [44], [9], [63], [3, 13, 25, 35], [48, 69, 89, 97], [4,
      14, 26, 36], [12, 24], [66, 67], [49, 68, 70, 88], [11], [64], [10], [43], [51, 72, 74, 91], [50, 53,
       73, 98], [15, 27], [5, 37], [75, 90], [52, 77], [16], [76], [17], [78, 79], [55, 93], [18, 30], [6,
      38], [57, 80, 81, 94], [54, 56, 83, 99], [40], [32], [33], [61], [65], [8], [59], [85], [28], [92],
      [29]]
[3]. Individualize 7 for next round.

[4]. CCR→[[95], [0], [42], [19], [58], [1], [21], [41], [84], [22], [86], [23], [87, 46], [2, 34], [96, 71,
       45, 47], [60], [20], [62], [44], [9], [63], [3, 35, 25, 13], [48, 97, 69, 89], [4, 36, 26, 14], [24,
       12], [66, 67], [49, 68, 70, 88], [11], [64], [10], [43], [51, 72, 74, 91], [27, 15], [90, 75], [28],
       [92], [29], [55], [6], [54], [40], [32], [33], [61], [65], [8], [59], [85], [7], [82], [80], [38],
      [93], [99], [39], [83], [31], [56], [94], [30], [79], [81], [17], [76], [78], [18], [57], [16], [52,
      77], [5, 37], [50, 98, 53, 73]]
[4]. SYM→[[95], [0], [42], [19], [58], [1], [21], [41], [84], [22], [86], [23], [46, 87], [2, 34], [45, 47,
       71, 96], [60], [20], [62], [44], [9], [63], [3, 13, 25, 35], [48, 69, 89, 97], [4, 14, 26, 36], [12,
       24], [66, 67], [49, 68, 70, 88], [11], [64], [10], [43], [51, 72, 74, 91], [15, 27], [75, 90], [28],
       [92], [29], [55], [6], [54], [40], [32], [33], [61], [65], [8], [59], [85], [7], [82], [80], [38],
      [93], [99], [39], [83], [31], [56], [94], [30], [79], [81], [17], [76], [78], [18], [57], [16], [52,
      77], [5, 37], [50, 53, 73, 98]]
[4]. Individualize 46 for next round.

[5]. CCR→[[95], [0], [42], [19], [58], [1], [21], [41], [84], [22], [86], [23], [87], [2], [45, 47], [60],
      [20], [62], [44], [9], [63], [3, 25], [48, 89], [4, 26], [24, 12], [66, 67], [49, 88], [11], [64],
      [10], [43], [51, 74], [27, 15], [90, 75], [28], [92], [29], [55], [6], [54], [40], [32], [33], [61],
      [65], [8], [59], [85], [7], [82], [80], [38], [93], [99], [39], [83], [31], [56], [94], [30], [79],
      [81], [17], [76], [78], [18], [57], [16], [52], [5], [50, 53], [46], [34], [96, 71], [35, 13], [97,
      69], [68, 70], [36, 14], [72, 91], [98, 73], [37], [77]]
[5]. SYM→[[95], [0], [42], [19], [58], [1], [21], [41], [84], [22], [86], [23], [87], [2], [45, 47], [60],
      [20], [62], [44], [9], [63], [3, 25], [48, 89], [4, 26], [12, 24], [66, 67], [49, 88], [11], [64],
      [10], [43], [51, 74], [15, 27], [75, 90], [28], [92], [29], [55], [6], [54], [40], [32], [33], [61],
      [65], [8], [59], [85], [7], [82], [80], [38], [93], [99], [39], [83], [31], [56], [94], [30], [79],
       [81], [17], [76], [78], [18], [57], [16], [52], [5], [50, 53], [46], [34], [71, 96], [13, 35], [69,
       97], [68, 70], [14, 36], [72, 91], [73, 98], [37], [77]]
[5]. Individualize 45 for next round.

Calculation completed after 5 rounds.
Discrete coloring→[[95], [0], [42], [19], [58], [1], [21], [41], [84], [22], [86], [23], [87], [2], [47],
       [60], [20], [62], [44], [9], [63], [3], [48], [4], [12], [66], [49], [11], [64], [10], [43], [51],
       [27], [90], [28], [92], [29], [55], [6], [54], [40], [32], [33], [61], [65], [8], [59], [85], [7],
       [82], [80], [38], [93], [99], [39], [83], [31], [56], [94], [30], [79], [81], [17], [76], [78], [18],
        [57], [16], [52], [5], [50], [46], [34], [96], [35], [97], [68], [36], [91], [98], [37], [77], [45],
        [25], [89], [88], [24], [67], [70], [13], [71], [69], [14], [72], [73], [15], [74], [75], [26],
       [53]]

Certificate→[[41, 42], [2, 7, 40], [1, 41], [49, 55, 61], [48, 54], [18, 30, 40], [7, 8, 47], [1, 6], [6,
       9], [8, 10, 20], [9, 11], [10, 12, 71], [11, 72], [14, 71, 82], [13, 83], [16, 45], [15, 17, 18],
       [16, 19], [5, 16], [17, 20, 46], [9, 19], [22, 26, 82], [21, 23], [22, 31, 70], [25, 26, 76], [24,
       27], [21, 24], [25, 28, 87], [27, 29], [28, 30, 44], [5, 29], [23, 32], [31, 33, 78], [32, 34], [33,
       35, 97], [34, 36], [35, 37, 52], [36, 38], [37, 39, 57], [38, 48], [1, 5], [0, 2, 47], [0, 43, 44],
       [42, 45], [29, 42], [15, 43, 46], [19, 45], [6, 41], [4, 39, 66], [3, 56], [54, 65], [52, 53, 55],
       [36, 51], [51, 54], [4, 50, 53], [3, 51], [49, 57, 58], [38, 56], [56, 59], [58, 60, 61], [59, 62],
       [3, 59], [60, 63, 64], [62, 67], [62, 65], [50, 64, 66], [48, 65], [63, 68, 81], [67, 69], [68, 70,
       99], [23, 69], [11, 13], [12, 73, 90], [72, 74], [73, 75, 76], [74, 77], [24, 74], [75, 78, 79], [32,
        77], [77, 80], [79, 81, 94], [67, 80], [13, 21], [14, 84, 85], [83, 98], [83, 86], [85, 87, 88],
       [27, 86], [86, 89], [88, 90, 91], [72, 89], [89, 92], [91, 93, 94], [92, 95], [80, 92], [93, 96, 97],
        [95, 98], [34, 95], [84, 96, 99], [69, 98]]
```

## *B.2. Strongly Regular Graphs*

A *strongly regular* graph $G$ is a simple graph defined by the tuple $(n, d, l, m)$ where $n$ is the number of vertices, $d$ is the number of neighbors for each $u \in V$, $l$ is the number of common neighbors for each pair of adjacent vertices and $m$ the number of common neighbors for each pair of non-adjacent vertices.

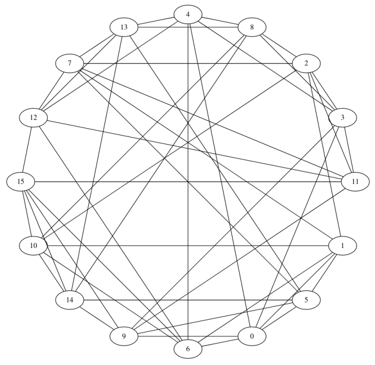


Figure 4: Strongly regular graph $G$ of the family (16,6,2,2).

Listing 2: Application of Master Algorithm on Strongly regular graph $G$

```
[0]. Subdivision of input graph computed.
[0]. Ranking of open neighborhood subgraphs computed.

[1]. CCR→[[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63], [0,
       1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]]
[1]. SYM→[[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63], [0,
       1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]]
[1]. Individualize 16 for next round.

[2]. CCR→[[17, 22], [0, 1], [16], [8, 11], [60, 61, 62, 63], [2, 3], [5, 6], [19, 20, 23, 24], [42, 43, 45,
        46], [12, 13, 14, 15], [4, 7, 9, 10], [18, 21, 25, 26], [48, 49, 38, 54, 39, 55, 56, 57], [36, 40,
       41, 44], [51, 52, 58, 59], [50, 37, 53, 47], [27], [32, 34, 28, 30], [33, 35, 29, 31]]
[2]. SYM→[[17, 22], [0, 1], [16], [8, 11], [60, 63], [61, 62], [2, 3], [5, 6], [19, 20, 23, 24], [42, 43,
       45, 46], [12, 13, 14, 15], [4, 7, 9, 10], [18, 21, 25, 26], [38, 49, 54, 57], [39, 48, 55, 56], [36,
       40, 41, 44], [51, 52, 58, 59], [37, 47, 50, 53], [27], [28, 30, 32, 34], [29, 31, 33, 35]]
```

```
[2]. Individualize 17 for next round.

[3]. CCR→[[22], [0], [16], [8, 11], [60, 61, 62, 63], [3], [5, 6], [19, 20], [42, 43, 45, 46], [12, 13, 14,
       15], [4, 9], [18, 21], [38, 54, 39, 55], [36, 41], [51, 52, 58, 59], [37, 53], [27], [32, 34], [33,
      35], [17], [1], [2], [28, 30], [29, 31], [7, 10], [25, 26], [48, 49, 56, 57], [40, 44], [50, 47],
      [23, 24]]
[3]. SYM→[[22], [0], [16], [8, 11], [60, 63], [61, 62], [3], [5, 6], [19, 20], [42, 46], [43, 45], [13,
      15], [12, 14], [4, 9], [18, 21], [38, 54], [39, 55], [36, 41], [52, 58], [51, 59], [37, 53], [27],
      [32, 34], [33, 35], [17], [1], [2], [28, 30], [29, 31], [7, 10], [25, 26], [49, 57], [48, 56], [40,
      44], [47, 50], [23, 24]]
[3]. Individualize 8 for next round.

Calculation completed after 3 rounds.
Discrete coloring→[[22], [0], [16], [11], [60], [3], [6], [20], [45], [12], [4], [18], [38], [36], [51],
      [37], [27], [32], [33], [17], [1], [2], [30], [29], [10], [26], [57], [44], [50], [24], [8], [58],
      [53], [35], [31], [47], [7], [28], [25], [48], [40], [5], [19], [42], [41], [23], [9], [21], [55],
      [34], [15], [61], [63], [46], [52], [59], [43], [14], [56], [54], [62], [13], [39], [49]]

Certificate→[[20, 21], [2, 7, 11, 19, 42, 47], [1, 20], [31, 32, 33, 34, 35, 55], [9, 61], [16, 17, 18, 19,
       33, 49], [7, 8, 13, 27, 29, 53], [1, 6], [6, 9], [4, 8, 12, 31, 39, 51], [11, 12, 13, 15, 17, 62],
      [1, 10], [9, 10], [6, 10], [30, 61], [10, 30], [5, 21], [5, 10], [5, 30], [1, 5], [0, 2, 25, 29, 38,
      45], [0, 16, 22, 23, 34, 37], [21, 24], [21, 30], [22, 25, 26, 27, 28, 58], [20, 24], [24, 50], [6,
      24], [24, 30], [6, 20], [14, 15, 18, 23, 28, 54], [3, 9], [3, 46], [3, 5], [3, 21], [3, 36], [35, 37,
       38, 39, 40, 63], [21, 36], [20, 36], [9, 36], [36, 41], [40, 42, 43, 44, 45, 56], [1, 41], [41, 61],
       [41, 46], [20, 41], [32, 44, 47, 48, 49, 59], [1, 46], [46, 50], [5, 46], [26, 48, 51, 52, 53, 55],
      [9, 50], [50, 57], [6, 50], [30, 57], [3, 50], [41, 57], [52, 54, 56, 58, 59, 60], [24, 57], [46,
      57], [57, 61], [4, 14, 43, 60, 62, 63], [10, 61], [36, 61]]
```

## B.3. Frucht Graph

A Frucht Graph [8] has no symmetric vertices, thus possesses only the trivial automorphism (identity or rigid graph).

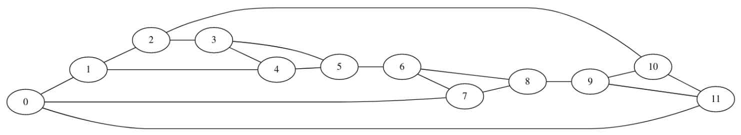


Figure 5: Frucht graph $G$

Listing 3: Application of Master Algorithm on Frucht graph $G$

```
[0]. Subdivision of input graph computed.
[0]. Ranking of open neighborhood subgraphs computed.

[1]. CCR→[[12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], [0, 1, 2, 3, 4, 5, 6,
      7, 8, 9, 10, 11]]
[1]. SYM→[[19], [21], [20], [24], [26], [17], [15], [13], [12], [4], [3], [9], [8], [6], [2], [1], [0],
      [5], [7], [25], [23], [18], [10], [29], [27], [11], [28], [14], [22], [16]]
[1]. Individualize 19 for next round.

Calculation completed after 1 rounds.
Discrete coloring→[[16], [4], [19], [11], [29], [1], [5], [21], [22], [6], [23], [7], [13], [25], [0],
      [14], [10], [2], [17], [15], [3], [20], [27], [18], [9], [26], [28], [8], [24], [12]]

Certificate→[[1, 5], [0, 2, 7], [1, 20], [4, 15, 26], [3, 16], [0, 19, 29], [7, 8, 21], [1, 6], [6, 9], [8,
       10, 28], [9, 11], [10, 12, 13], [11, 14], [11, 27], [12, 15, 29], [3, 14], [4, 22, 23], [18, 19,
      23], [17, 20], [5, 17], [2, 18, 21], [6, 20], [16, 24], [16, 17], [22, 25, 26], [24, 27], [3, 24],
      [13, 25, 28], [9, 27], [5, 14]]
```

## B.4. Bipartite Graphs

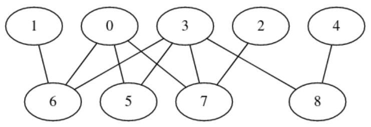


Figure 6: Bipartite graph $G$

Listing 4: Application of Master Algorithm on bipartite graph $G$

```
[0]. Graph ([0, 1, 2, 3, 4, 5, 6, 7, 8], [{0,5}, {0,6}, {0,7}, {1,6}, {2,7}, {3,5}, {3,6}, {3,7}, {3,8},
     {4,8}]) already bipartite.
[0]. Ranking of open neighborhood subgraphs computed.

[1]. CCR→[[4], [5], [0], [3], [6, 7], [1, 2], [8]]
[1]. SYM→[[4], [5], [0], [3], [6, 7], [1, 2], [8]]
[1]. Individualize 6 for next round.

Calculation completed after 1 rounds.
Discrete coloring→[[4], [5], [0], [3], [7], [1], [8], [6], [2]]

Certificate→[[6], [2, 3], [1, 4, 7], [1, 4, 6, 7], [2, 3, 8], [7], [0, 3], [2, 3, 5], [4]]
```

## References

[1] C. Berkholz, P. Bonsma, M. Grohe, Tight lower and upper bounds for the complexity of canonical colour refinement, Theory of Computing Systems 60 (4) (2017) 581–614. doi:10.1007/s00224-016-9686-0.
URL https://doi.org/10.1007/s00224-016-9686-0

[2] M. Grohe, K. Kersting, M. Mladenov, P. Schweitzer, Color refinement and its applications, 2021.
URL https://api.semanticscholar.org/CorpusID:59069015

[3] K. S. Booth, C. J. Colbourn, Problems polynomially equivalent to graph isomorphism, Computer Science Department, Univ., 1979.

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

[5] L. Babai, Graph isomorphism in quasipolynomial time [extended abstract], in: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, STOC '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 684–697. doi:10.1145/2897518.2897542.
URL https://doi.org/10.1145/2897518.2897542

[6] T. Miyazaki, The complexity of McKay's canonical labeling algorithm, in: Groups and computation II. Workshop on groups and computation, June 7–10, 1995, New Brunswick, NJ, USA, Providence, RI: American Mathematical Society, 1997, pp. 239–256.

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

[8] R. Frucht, Graphs of degree three with a given abstract group, Canadian Journal of Mathematics 1 (4) (1949) 365–378. doi:10.4153/CJM-1949-033-6.