Contemporary Mathematics

# McKay's Canonical Graph Labeling Algorithm

Stephen G. Hartke and A. J. Radcliffe

Abstract. The problem of deciding whether two graphs are isomorphic is fundamental in graph theory. Moreover, the flexibility with which other combinatorial objects can be modeled by graphs has meant that efficient programs for deciding whether graphs are isomorphic have also been used to study a variety of other combinatorial structures. Not only is the graph isomorphism problem a very practical one, it is also fascinating from a complexity-theoretic point of view. Graph isomorphism is one of the few problems that are clearly in NP but not known either to be solvable in polynomial time, or to be NP-complete.

Various people have worked to create algorithms for graph isomorphism which are "practical in practice". One of the most powerful and best known of these algorithms is due to Brendan McKay. It is known that his algorithm has exponential running time on some inputs, but it performs exceptionally well under most circumstances. In this article we aim to provide an introduction to the essential ideas of McKay's algorithm.

**Keywords:** graph isomorphism, graph automorphism, canonical labeling, canonical isomorph, nauty

## 1. Introduction

The problem of deciding whether two graphs are isomorphic is fundamental in graph theory. Moreover, the flexibility with which other combinatorial objects can be modeled by graphs has meant that efficient programs for deciding whether graphs are isomorphic have also been used to study and enumerate (see [**16**]) a variety of other combinatorial structures, including distance-regular graphs [**5**], strongly regular graphs [**7**], block designs [**4**], Latin squares [**18**], partial geometries [**24**], and integer programs [**21**].

Not only is the graph isomorphism problem (which we abbreviate GI henceforth) a very practical one, it is also fascinating from a complexity-theoretic point of view. It is one of the few problems that are clearly in NP but not known either to be solvable in polynomial time, or to be NP-complete[1].

The first author is partially supported by a Maude Hammond Fling Faculty Research Fellowship from the University of Nebraska Research Council.

[1]GI is in NP since an explicit isomorphism serves as a certificate that two graphs are isomorphic.

There is a large amount of heuristic evidence that GI is not NP-complete. For example GI behaves markedly differently from problems that are known to be NP-complete, in that seemingly mild constraints on the problem allow polynomial time algorithms. GI is polynomial time for: graphs of bounded degree [**13**], graphs of bounded genus [**8, 19**], graphs of bounded eigenvalue multiplicity [**2**], and graphs of bounded treewidth [**3**]. Contrast this behavior with that of, say, the graph coloring problem, which is NP-complete even for the special case of deciding 3-colorability of planar graphs of maximum degree at most four [**10**]. Even more strikingly, the problem of counting the number of isomorphisms between two given graphs is Turing reducible to GI itself [**14**].[2] Compare this to the problem of determining whether a bipartite graph has a perfect matching. This is a polynomial time decision problem, yet the corresponding counting problem is complete for #P (the class of counting problems corresponding to decision problems in NP) [**25**].

There is further evidence that the problem of graph isomorphism is not NP-complete, in that if it were the polynomial hierarchy (of complexity classes "above" NP and co-NP) would collapse[3]. For more details about the complexity issues around GI, see the lovely book by Köbler, Schöning, and Torán [**12**].

Despite this evidence that GI is in some ways "easy", there is no known polynomial time algorithm for it. GI is known to have time complexity at most $\exp\left(O(n^{2/3})\right)$ for graphs with $n$ vertices [**1**]. Various people have worked to create general algorithms for GI which are "practical in practice". One of the most powerful and best known of these algorithms is due to Brendan McKay [**15**]. It is known that his algorithm has exponential running time on some inputs [**9, 20**], but in general it performs exceptionally well. McKay has implemented his algorithm in the software package `nauty` (No AUTomorphisms, Yes?), freely available at his website [**17**]. R. L. Miller has implemented the algorithm as the `NICE` (NICE Isomorphism Check Engine) module of the open-source SAGE mathematics software [**23**].

In this article we aim to provide an introduction to the essential ideas of McKay's algorithm. There are three main strands to the `nauty` algorithm:

- using, iteratively, degree information;
- building a search tree examining choices not determined by degree information; and
- using graph automorphisms, as they are found, to prune the search tree.

We introduce all three strands in Section 2 and discuss their connections. Later, in Sections 4, 5, and 6, we go into more detail about how they are incorporated into McKay's algorithm. In Section 7 we discuss additional aspects which, although useful in certain cases, do not affect the main thrust of the algorithm.

---

[2] I.e., if we are allowed to use GI as a constant time subroutine then the number of isomorphisms between two given graphs can be determined in polynomial time.

[3] The polynomial hierarchy consists of two inductively defined sequences of complexity classes $\Sigma_k^P$ and $\Pi_k^P$, $k \geq 0$. (The superscript $P$ stands for the complexity class $P$.) The sequences start with $\Sigma_0^P = \Pi_0^P =$ P. Then we say that a problem is in $\Sigma_k^P$ if every instance (for which the answer is "yes") has some certificate (of length bounded by some polynomial in the length of the instance) such that the problem of checking the certificate is in $\Pi_{k-1}^P$. Then we let $\Pi_k^P =$ co-$\Sigma_k^P$. For instance $\Sigma_1^P = NP$, $\Pi_1^P =$co-NP, and a problem is in $\Sigma_2^P$ if it has a certification scheme for which the certificate-checking problem is in co-NP. It is widely believed, though not proven, that all these classes are distinct.

## 2. Motivation

We start by noting that we can assume, without loss of generality, that all the graphs we consider have vertex set $[n] = \{1, 2, \ldots, n\}$. We will only consider simple undirected graphs, so we can identify a graph with its edge set, considered as a subset of $\binom{[n]}{2}$, the set of all unordered pairs from $[n]$.

One natural approach to the graph isomorphism problem is the use of canonical isomorphs: picking a canonical representative from each isomorphism class of graphs on $[n]$. Then testing isomorphism between two graphs reduces to checking equality of their corresponding canonical isomorphs. Note that equality of graphs is easy to test; the real work is performed in computing the canonical isomorph.

DEFINITION 1. An *isomorph* of a graph $G$ is a graph with vertex set $[n]$ that is isomorphic to $G$. A *canonical isomorph function* assigns to every graph $G$ an isomorph $C(G)$ such that whenever $H$ is isomorphic to $G$ we have $C(H) = C(G)$. We call $C(G)$ the *canonical isomorph* of $G$.

In the rest of the literature, including the title of this article, $C(G)$ is referred to as a "canonical labeling". However, we prefer the term "canonical isomorph" since it emphasizes the fact that $C(G)$ is a graph and not a labeling of $G$.

One way to define a canonical isomorphism function is to specify a total order $\leq$ on graphs on $[n]$. Then we define $C_{\leq}(G)$ to be the $\leq$-largest graph in the isomorphism class of $G$. Clearly this satisfies the conditions for a canonical isomorph function. One such example is the canonical isomorph function $C_{\preceq}$ defined by Read [**22**] and independently by Faradžev [**6**]. The ordering $\preceq$ they use is a lexicographic total order induced by a fixed total ordering on $\binom{[n]}{2}$. Represent a graph $G$ by the binary sequence $i(G)$ of length $\binom{n}{2}$ in which the $j$-th entry is 1 exactly if the $j$-th unordered pair (in our fixed ordering) is an edge in $G$. Then $G \preceq H$ if $i(G) \leq i(H)$ lexicographically.

Unfortunately $C_{\preceq}$ has the drawback of being difficult to compute. Typically there is no alternative to checking essentially every permutation of the vertices to see whether it produces the graph that is $\preceq$-greatest.

One reason for the weakness of this sort of approach is that it does not exploit any graph theoretical information. How might one use such information to pick out "landmarks" in the graph? One starting point might be to have vertices in the canonical isomorph appear in increasing order of degree. Clearly this is not sufficient to determine the canonical isomorph. However, this local information can then be propagated around the graph. For instance, if there is a unique vertex $v$ of some particular degree, then the neighbors of $v$ can be distinguished from the non-neighbors. Iterating this idea, the second neighborhood of $v$ can then be distinguished, and so on. The first main strand of McKay's algorithm is to distinguish vertices according to degree, and then to propagate this local information using the process described above. We discuss this propagation scheme in Section 4.

Unfortunately, it is quite possible that the propagation of local information starting from degrees will stabilize without all of the vertices being distinguished. For instance, if $G$ is regular, then the process does not even start. Using stronger local information than degrees might provide more information in certain cases, at the cost of more computation. However, in many instances no local information is useful. Indeed, in vertex-transitive graphs, *no* vertex can be distinguished by any information, local or global.

McKay's algorithm proceeds at this point by introducing artificial asymmetry. Faced with a set $S$ of (to this point) indistinguishable vertices, we distinguish, in turn, each vertex of $S$. This information is then propagated to the rest of the graph. In this way, we build a search tree (examining all choices) whose leaves are isomorphs of $G$. McKay's canonical isomorph $C_M$ is the $\preceq$-greatest of these isomorphs appearing at leaves. Thus, in comparison to $C_{\preceq}$, the algorithm uses graph theoretical information to reduce the number of candidate isomorphs from which the canonical isomorph is chosen. We discuss this aspect of McKay's algorithm in Section 5.

The final major aspect of McKay's algorithm is the use of automorphisms that are discovered while exploring the search tree to prune later computation. This allows us to compute $C_M$ more efficiently. This topic is covered in Section 6.

Before we discuss the details of McKay's canonical isomorph function and its implementation in `nauty` we introduce the notation we will use for group actions and isomorphisms.

## 3. Group Actions and Isomorphism

In order to talk about isomorphism carefully it is helpful to briefly discuss group actions.

Definition 2. An *action*[4] of a group $\Gamma$ on a set $X$ is a function from $X \times \Gamma$ to $X$, mapping $(x, g)$ to $x^g$, that satisfies the following conditions:

- for all $x \in X$ and $g, h \in \Gamma$ we have $(x^g)^h = x^{(gh)}$; and
- If $e$ is the identity of $\Gamma$ then $x^e = x$ for all $x \in X$.

For us the most important examples are various actions of $\Sigma_n$, the group of all permutations of $[n]$. We have chosen to write our group actions on the right; as a consequence we will also choose to compose our permutations on the right: for $\sigma, \gamma$ elements of $\Sigma_n$ the group product $\sigma\gamma$ is defined to be the permutation obtained by first applying $\sigma$ and then $\gamma$. With this convention $\Sigma_n$ acts on $[n]$, since for $\sigma, \gamma \in \Sigma_n$, $v \in [n]$, we have $(v^\sigma)^\gamma = v^{(\sigma\gamma)}$ (where, of course, $v^\sigma$ is the image of $v$ under $\sigma$).

Whenever $\Gamma$ acts on $X$ it also acts on subsets of $X$ simply by defining

$$A^g = \{x^g \,:\, x \in A\}\,.$$

Thus the natural action of $\Sigma_n$ on $[n]$ induces an action on $\binom{[n]}{2}$. As we have identified simple, undirected graphs with their edge sets, this action on pairs induces in turn an action of $\Sigma_n$ on the set of all graphs with vertex set $[n]$. This action defines the notion of isomorphism of graphs.

Definition 3. Two graphs $G$ and $H$ with vertex set $[n]$ are *isomorphic* if there exists a permutation $\gamma \in \Sigma_n$ such that $H = G^\gamma$.

Similarly, if $\Gamma$ acts on $X$, then it also acts on sequences $(x_i)_{i=1}^k$ from $X$ by

$$\big((x_i)_1^k\big)^\gamma = (x_i^\gamma)_1^k.$$

[4]To be a little more careful, what we define here is called a *right action* of $\Gamma$ on $X$.

## 4. Propagating Degree Information

The first aspect of McKay's algorithm we will focus on is the propagation of degree information. We describe a classification of the vertices of $G$ using an ordered partition. From some such partitions we can deduce further distinctions; otherwise, we call the ordered partition *equitable*. We first give the appropriate definitions and then describe the procedure in McKay's algorithm which refines an ordered partition until it becomes equitable.

DEFINITION 4. An *ordered partition* $\pi$ of $[n]$ is a sequence $(V_1, V_2, \ldots, V_r)$ of nonempty subsets of $[n]$ such that $\{V_1, V_2, \ldots, V_r\}$ is a partition of $[n]$. The subsets $V_1, V_2, \ldots, V_r$ are called the *parts* of $\pi$. A *trivial part* is a part of size 1. A *discrete partition* only has trivial parts, while the *unit partition* $\mu$ only has one part, namely $[n]$. The *length* of an ordered partition $\pi$ is the number of parts in $\pi$.

Ordered partitions come equipped with the natural partial order of refinement. Given two ordered partitions $\pi_1$ and $\pi_2$, we say that $\pi_1$ is *finer* than $\pi_2$ if:

- every part $V_i$ of $\pi_1$ is contained in a part $W_k$ of $\pi_2$, and
- earlier parts of $\pi_1$ are contained in earlier parts of $\pi_2$; *i.e.*, if $V_i$ and $V_j$ are parts of $\pi_1$ with $i \leq j$, and $W_k$ and $W_\ell$ are parts of $\pi_2$ such that $V_i \subseteq W_k$ and $V_j \subseteq W_\ell$, then $k \leq \ell$. 粗粒度

If $\pi_1$ is finer than $\pi_2$, then $\pi_2$ is *coarser* than $\pi_1$. (Note that "finer than" is not strict: it includes the case of equality.) The set of ordered partitions of $[n]$ with the relation "finer than" forms a partially ordered set whose unique maximal element is the unit partition $\mu$ and whose minimal elements are the discrete ordered partitions of $[n]$.

Suppose now that the ordered partition $\pi$ encodes a classification of the vertices. Vertices in different parts of $\pi$ have already been distinguished from each other; vertices in the same part have not. If two vertices $v, w$ belong to the same part of $\pi$ but have different degrees into a part of $\pi$ then we can distinguish $v$ from $w$, refining our classification. Our next definition describes the situation in which no further propagation of degree information is possible.

DEFINITION 5. An ordered partition $\pi = (V_1, V_2, \ldots, V_r)$ of $[n]$ is an *equitable ordered partition* (with respect to $G$) if, for all $1 \leq i, j \leq r$, $\deg(v, V_j) = \deg(w, V_j)$ for all $v, w \in V_i$. An equitable ordered partition $\tau$ is a *coarsest equitable refinement*[5] *of* $\pi$ if $\tau$ is finer than $\pi$ and there is no equitable ordered partition finer than $\pi$ and strictly coarser than $\tau$.

Thus, for example, in a regular graph the unit partition is equitable. Similarly, in a strongly regular graph the ordered partition

$$\left(\{v\}, N(v), [n] \setminus (\{v\} \cup N(v))\right)$$

(where $N(v)$ is the neighborhood of $v$) is equitable for any vertex $v$.

If $\pi$ is an inequitable ordered partition we would like to extract, iteratively, all the consequences (and *only* the consequences) of the distinctions expressed by $\pi$. This amounts to finding a coarsest equitable refinement of $\pi$. In the remainder

[5]Despite the name, coarsest equitable refinements are not necessarily unique. A better name might be "maximal equitable refinements", but for reasons both of history and clarity we prefer McKay's terminology. However, coarsest equitable refinements *are* unique up to ordering of the parts; see Lemma 7.

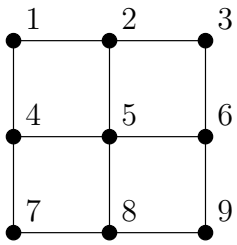


FIGURE 1. Example graph $G$ on 9 vertices.

of this section we present the equitable refinement procedure used in McKay's algorithm[6]. The procedure accepts as input a graph $G$ and an ordered partition $\pi$ of $[n]$, and returns a coarsest equitable refinement $R(\pi)$ of $\pi$.

DEFINITION 6. Given an inequitable ordered partition $\pi = (V_1, V_2, \ldots, V_r)$, we say that $V_j$ *shatters* $V_i$ if there exist two vertices $v, w \in V_i$ such that $\deg(v, V_j) \neq \deg(w, V_j)$. The *shattering of* $V_i$ *by* $V_j$ is the ordered partition $(X_1, X_2, \ldots, X_t)$ of $V_i$ such that if $v \in X_k$ and $w \in X_\ell$ then $k < \ell$ if and only if $\deg(v, V_j) < \deg(w, V_j)$. Thus, $(X_1, X_2, \ldots, X_t)$ sorts the vertices of $V_i$ by their degree to $V_j$.

Equitable Refinement Procedure.

*Input*: An unordered simple graph $G$ with vertex set $[n]$, and an ordered partition $\pi$ of $[n]$.

*Output*: An ordered partition $R(\pi)$.

*Initialize*: Let $\tau$ be the ordered partition $\pi$.

*Iterate*: If $\tau = (V_1, V_2, \ldots, V_r)$, then let $B = \{(i, j) : V_j \text{ shatters } V_i\}$. If $B$ is empty, then stop, reporting $\tau$ as the output $R(\pi)$. Otherwise, let $(i, j)$ be the minimum element of $B$ under the lexicographic order. Let $(X_1, X_2, \ldots, X_t)$ be the shattering of $V_i$ by $V_j$. Replace $\tau$ by the ordered partition where $V_i$ is replaced by $X_1, X_2, \ldots, X_t$; that is, replace $\tau = (V_1, V_2, \ldots, V_r)$ with

$$(V_1, V_2, \ldots, V_{i-1}, X_1, X_2, \ldots, X_t, V_{i+1}, \ldots, V_r).$$

组内都是等度

EXAMPLE. Let $G$ be the graph on 9 vertices shown in Figure 1. We show the execution of the equitable refinement procedure on two ordered partitions of [9], namely the unit partition $\mu$ and another partition.

| $\pi$ | $B$ | $V_i$ | $V_j$ |
|---|---|---|---|
| (1 2 3 4 5 6 7 8 9) | $\{(1,1)\}$ | (123456789) | (123456789) |
| (1 3 7 9 \| 2 4 6 8 \| 5) | $\emptyset$ | | |

{1379} 到 {123456789} 的度为2
{2468} 到 {123456789} 的度为3
{5} 到 {123456789} 的度为4

| $\pi$ | $B$ | $V_i$ | $V_j$ |
|---|---|---|---|
| (1 \| 3 7 9 \| 2 4 6 8 \| 5) | $\{(3,1),(3,2)\}$ | (2468) | (1) |
| (1 \| 3 7 9 \| 6 8 \| 2 4 \| 5) | $\{(2,4)\}$ | (379) | (24) |
| (1 \| 9 \| 3 7 \| 6 8 \| 2 4 \| 5) | $\emptyset$ | | |

{2468} 被 {1} 分割成 {24}，{68}，
因为 {24} 与1邻接，{68} 与1不邻接
所以 {68} 排在 {24} 前面

[6]McKay's actual equitable refinement procedure differs slightly in the order in which parts are chosen to be shattered. However, only two properties of the procedure are necessary. Firstly, it must produce a coarsest equitable refinement of $\pi$. Secondly, the refinement $R(\pi)$ must satisfy $R(\pi^\gamma) = R(\pi)^\gamma$ for any $\gamma \in \Sigma_n$.

We prove, in Proposition 8 below, the correctness of the equitable refinement procedure. Our proof relies on the following lemma which is just a restatement of the fact (which is straightforward to prove) that in the lattice of *unordered* partitions, there is a unique unordered coarsest equitable refinement of any partition.

LEMMA 7. *Let $\pi$ be an ordered partition of $[n]$. Suppose that $\zeta$ and $\xi$ are coarsest equitable refinements of $\pi$. Then $\zeta$ and $\xi$ have the same parts and differ only in the order of the parts.*

PROPOSITION 8. *The partition $R(\pi)$ returned by the equitable refinement procedure is a coarsest equitable refinement of $\pi$.*

PROOF. At each iteration of the procedure, some part $V_i$ of $\tau$ is shattered, producing a new ordered partition whose length is greater than that of $\tau$. Since every discrete partition is equitable, the length of $\tau$ is at most $n$. Hence the number of iterations is bounded, and the equitable refinement procedure terminates.

The algorithm halts only when an equitable ordered partition $R(\pi)$ is obtained. Since $\tau$ at each stage is finer than that of the previous iteration, and hence is finer than $\pi$, the partition $R(\pi)$ is also finer than $\pi$. That $R(\pi)$ is a coarsest equitable refinement follows by a straightforward argument using Lemma 7. □

我fo了，车轱辘话:
因为R(π)是π的更好分割，所以R(π)的分割比π好
？？？

## 5. The Search Tree

McKay's algorithm starts by forming the equitable refinement of the unit partition, thereby extracting all of the initial degree information. Having reached an equitable partition, we need to introduce artificial distinctions between vertices. However, we must be careful to examine all relevant choices. We systematically explore the space of equitable ordered partitions using a search tree. The next definition describes the way we make these artificial distinctions, forming children in the search tree.

至少一个

DEFINITION 9. Let $\pi$ be an equitable ordered partition of $[n]$ with a nontrivial part $V_i$, and let $u \in V_i$. The *splitting of $\pi$ by $u$*, denoted by $\pi \perp u$, is the equitable refinement $R(\pi')$ of the ordered partition $\pi' = (V_1, V_2, \ldots, \{u\}, V_i \setminus \{u\}, V_{i+1}, \ldots, V_r)$. (Note that $\pi \perp u$ is strictly finer than $\pi$.)

Crucially, the children of an equitable ordered partition in the search tree do not correspond to all possible splittings of $\pi$. When we artificially distinguish a vertex $u$, the only alternatives we need to consider are vertices indistinguishable (so far) from $u$, i.e., vertices in the same part of $\pi$. At each stage we chose to split the first non-trivial part[7] of $\pi$.

We record in the search tree not only the current equitable ordered partition, but also the sequence of vertices used for splittings.

[7]To reduce the branching factor, McKay actually chooses the first *smallest* part of $\pi$. The method of choosing the part for splitting $\pi$ is irrelevant as long as it is an isomorphism invariant of unordered partitions. That is to say, that if the $i$-th part of $\pi$ is chosen, then also the $i$-th part of $\pi^\gamma$ is chosen for any $\gamma \in \Sigma_n$.

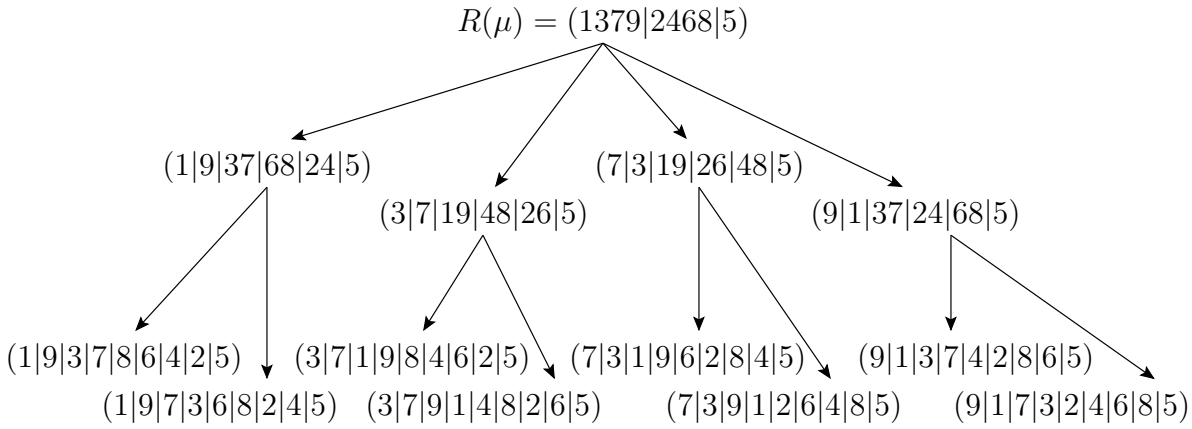


FIGURE 2. The search tree $T(G)$ for the graph $G$ of Figure 1. Only the ordered partitions associated with each node of the search tree are shown.

DEFINITION 10. The *search tree* $T(G)$ is the rooted tree whose nodes

$$\Big\{(\pi;\underset{\sim}{u}) : \pi \text{ is an ordered partition of } [n];\ \underset{\sim}{u} = (u_1, u_2, \ldots, u_k) \text{ is a sequence of distinct vertices;}$$
$$\pi = (\ldots(R(\mu) \perp u_1) \perp u_2)\ldots) \perp u_k;$$
$$u_i \text{ is in the first non-trivial part of } (\ldots(R(\mu) \perp u_1) \perp u_2)\ldots) \perp u_{i-1} \text{ for each } i. \Big\}.$$

We allow the sequence $\underset{\sim}{u}$ to be empty, in which case $\pi = R(\mu)$. This is in fact the root node of the tree. There is an arc in $T(G)$ directed from the node $(\pi;\underset{\sim}{u})$ to the node $(\tau;\underset{\sim}{v})$ if $\underset{\sim}{u}$ is a prefix of $\underset{\sim}{v}$ and $\text{length}(\underset{\sim}{v}) = \text{length}(\underset{\sim}{u}) + 1$.

The *terminal nodes* of the search tree are nodes with no outgoing arcs. These terminal nodes correspond to discrete ordered partitions of $[n]$. Given a discrete ordered partition $\pi = (V_1, V_2, \ldots, V_n)$, we define $\sigma_\pi \in \Sigma_n$ to be the permutation such that $i^{(\sigma_\pi)} = j$ if $V_j$ contains $i$. For example, if $\pi = (1|9|3|7|8|6|4|2|5)$, then $\sigma_\pi$ maps $1 \mapsto 1$, $9 \mapsto 2$, $3 \mapsto 3$, $7 \mapsto 4$, $8 \mapsto 5$, and so on. To each terminal node $p = (\pi;\underset{\sim}{u})$ we associate the isomorph $G^{(\sigma_\pi)}$ of $G$.

We can now completely describe McKay's canonical isomorph function.

DEFINITION 11. McKay's canonical isomorph function $C_M(G)$ is defined to be

$$C_M(G) = \max_{\preceq} \Big\{ G^{(\sigma_\pi)} : (\pi, \underset{\sim}{u}) \text{ is a leaf of } T(G) \Big\}.$$

We denote the set of isomorphs of $G$ appearing on the right hand side by $L(G)$.

EXAMPLE. The search tree $T(G)$ for the graph $G$ of Figure 1 is shown in Figure 2. For this example, all the isomorphs of $G$ associated to the terminal nodes are the same; this isomorph is the canonical isomorph $C_M(G)$. (So in particular $L(G)$ has size 1.) Figure 3 shows this graph $C_M(G)$. Thinking of the various $\sigma_\pi$'s as relabelings of $G$, we see many relabelings but only one isomorph. This is why we prefer the "canonical isomorph" terminology.

(1|9|3|7|8|6|4|2|5)
π_i = i

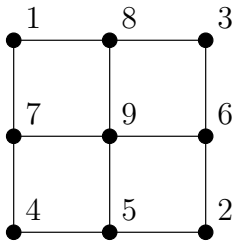


FIGURE 3. The canonical isomorph $C_M(G)$ for the graph $G$ of Figure 1.

We have an action of $\Sigma_n$ on sequences $\underset{\sim}{u}$ of vertices and on ordered partitions $\pi$. Thus we get an action on search trees, where the nodes of $T(G)^\gamma$ have the form $(\pi^\gamma; \underset{\sim}{u}^\gamma)$ for $(\pi; \underset{\sim}{u})$ in $T(G)$.

The following lemma contains the essence of the proof that $C_M(G)$ is a canonical isomorph function.

LEMMA 12. *If $H = G^\gamma$ for some $\gamma \in \Sigma_n$, then $T(H) = T(G)^\gamma$.*

PROOF. Note that the equitable refinement process respects the action of $\gamma$; that is, for an ordered partition $\pi$, $R(\pi^\gamma) = R(\pi)^\gamma$. Furthermore, if the first nontrivial part of $\pi$ is $V_i$, then the first nontrivial part of $\pi^\gamma$ is $(V_i)^\gamma$, which is the $i$-th part of $\pi^\gamma$. Therefore, the children of $(\pi; \underset{\sim}{u})^\gamma$ in $T(H)$ are the images under $\gamma$ of the children of $(\pi; \underset{\sim}{u})$ in $T(G)$. The result follows by induction on length$(\underset{\sim}{u})$. □

THEOREM 13. *$C_M(G)$ is a canonical isomorph function.*

PROOF. Note that in $T(G)$, the ordered partition $\pi$ associated to a node $m$ is strictly finer than the ordered partition associated to the parent of $m$. Hence, $T(G)$ is finite, $L(G)$ is finite, and the algorithm terminates. The output $C_M(G)$ is defined, by the algorithm, to be an isomorph of $G$.

By Lemma 12, for every terminal node $p = (\pi; \underset{\sim}{u})$ in $T(G)$, there exists a terminal node $p^\gamma = (\pi^\gamma; \underset{\sim}{u}^\gamma)$ in $T(H)$. From the action of $\gamma$ on the partition $\pi$, we have $\sigma_{(\pi^\gamma)} = \gamma^{-1}\sigma_\pi$. Hence,

$$H^{\sigma_{(\pi^\gamma)}} = H^{\gamma^{-1}\sigma_\pi} = (G^\gamma)^{\gamma^{-1}\sigma_\pi} = G^{(\sigma_\pi)},$$

and so the sets $L(G)$ and $L(H)$ of graphs are the same. Thus, the maxima are the same, and so $C_M(G) = C_M(H)$. □

## 6. Pruning the Search Tree through the use of Automorphisms

One reason that the search tree $T(G)$ might be large is if $G$ has a large automorphism group. In that case, the number of terminal nodes is at least the size of the automorphism group. However, the search tree is generated depth-first, and so automorphisms that are discovered during the search process can be used to *prune* the tree—discarding a section of the search tree not yet examined because it is known that no terminal node in that section will generate an isomorph of $G$ that is better than the ones already discovered.

If there exist two terminal nodes $p = (\pi; \underset{\sim}{u})$ and $t = (\tau; \underset{\sim}{v})$ where $G^{(\sigma_\pi)} = G^{(\sigma_\tau)}$, then $\sigma_\pi(\sigma_\tau)^{-1}$ is an automorphism of $G$, since

$$G^{\sigma_\pi(\sigma_\tau)^{-1}} = (G^{\sigma_\tau})^{(\sigma_\tau)^{-1}} = G.$$

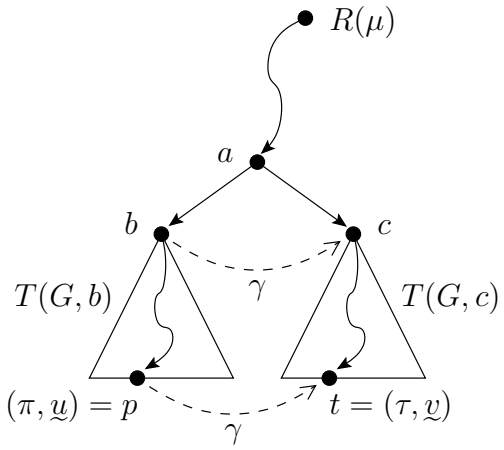


FIGURE 4. When an automorphism $\gamma$ is discovered, certain subtrees of the search tree are known to be isomorphic.

Let $a$ be the node in the search tree that is the deepest common ancestor of $p$ and $t$, and let $b$ be the child of $a$ that is an ancestor of $p$, and $c$ the child of $a$ that is an ancestor of $t$. (See Figure 4 for an illustration.) The permutation $\gamma = \sigma_\pi(\sigma_\tau)^{-1}$ sends $\pi$ to $\tau$, fixes $a$, and sends $b$ to $c$. For a node $m$ of the search tree, let $T(G,m)$ denote the subtree of $T(G)$ rooted at $m$. Then $T(G,c)$ is isomorphic to $T(G,b)$ via $\gamma$, and so the set of graphs generated from terminal nodes of $T(G,c)$ is the same as the set of graphs generated from terminal nodes of $T(G,b)$. The search tree $T(G)$ is examined depth-first so all of $T(G,b)$ is examined before $T(G,c)$, and there is no reason to examine any more of $T(G,c)$; the search can continue at $a$.

Automorphisms discovered during the search can also be used to prune the search tree in another way. Let $d$ be a node being re-visited by the depth-first search. Let $\Gamma$ be the group generated by the automorphisms discovered thus far, and let $\Phi$ be the subgroup of $\Gamma$ that fixes $d$. Suppose that $b$ and $c$ are children of $d$ where some element of $\Phi$ maps $b$ to $c$. If $T(G,b)$ has already been examined, then, as above, there is no need to examine $T(G,c)$. Hence $T(G,c)$ can be pruned from the tree[8].

EXAMPLE. The search tree $T(G)$ pruned through the use of automorphisms is shown in Figure 5. The search tree is examined depth-first by examining left children first. The automorphism $\gamma_1$ is discovered first, and at the root node, $\gamma_1$ has three orbits (1), (9), and (37) on the part (1379). Hence the child $c$ of the root node must still be examined. When the automorphism $\gamma_2$ is discovered, the remainder of $T(G,c)$ is immediately discarded. At the root node, the subgroup of $\mathrm{Aut}(G)$ generated by $\gamma_1$ and $\gamma_2$ has one orbit in (1379), and hence no more children of the root need to be examined.

Note that for every automorphism $\gamma \in \mathrm{Aut}(G)$, there exist two terminal nodes with associated discrete ordered partitions $\pi$ and $\tau$ such that $\gamma = \sigma_\pi(\sigma_\tau)^{-1}$. When

[8]In practice, only the generators of $\Gamma$ are stored and only those which fix $d$ are used to prune children of $d$. Moreover, the determination of which children to examine and which to prune is simplified by storing the orbits of this subgroup on $[n]$.

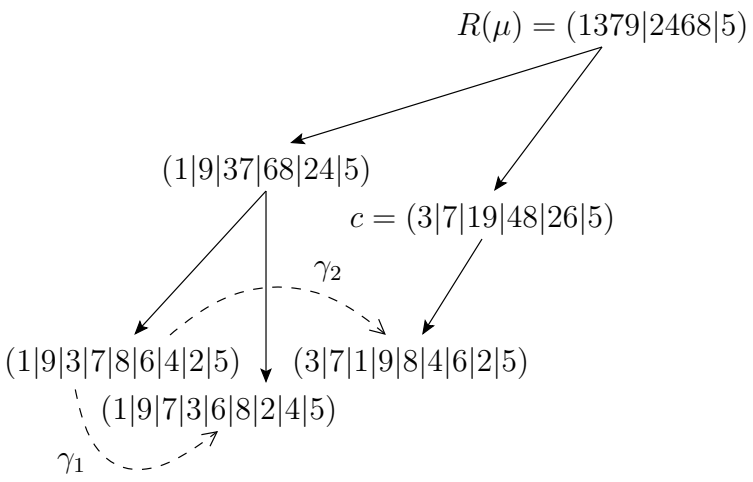


FIGURE 5. The search tree $T(G)$ pruned through the use of automorphisms.

pruning sections of the search tree, only the images of terminal nodes under the subgraph of $\mathrm{Aut}(G)$ generated thus far are removed. Thus, the canonical isomorph algorithm will find a complete set of generators for $\mathrm{Aut}(G)$.

EXAMPLE. The automorphism group of $G$ from Figure 1 is the dihedral group on 8 elements, which is generated by $\gamma_1$ (a reflection) and $\gamma_2$ (a rotation).

At this point, we have completed a description of the essential elements of McKay's canonical isomorph algorithm.

## 7. Additional Aspects

In [**15**], McKay describes several improvements to the basic canonical isomorph algorithm. Many of these improvements are primarily implementation details and do not change the fundamental mathematical concepts. For instance, McKay chooses a different part to shatter in the equitable refinement procedure, and a different part to split when forming children in the search tree. We have omitted these details for the sake of clarity. However, we mention here just a few of the further improvements that are not purely details of a particular implementation.

IMPROVEMENT 1. Sometimes automorphisms of $G$ can be determined at a non-terminal node $m = (\pi; \underset{\sim}{u})$ of the search tree. When this happens, they can be used to prune the search tree as before. For instance, this occurs when every nontrivial part of $\pi$ has size 2 or when $\pi$ has exactly one nontrivial part whose size is at most 5.

IMPROVEMENT 2. The search tree of the canonical isomorph algorithm can start with any equitable ordered partition, not just $R(\mu)$. With this modification, the canonical isomorph algorithm can be used to test for color-preserving isomorphisms of vertex-colored graphs.

IMPROVEMENT 3. It is possible to further restrict the number of candidates for the canonical isomorph by using a arbitrary function of pairs $(G, \pi)$ (where $\pi$ is an

ordered partition of $[n]$) that is constant on isomorphism classes. McKay calls such isomorphism invariants *indicator functions*. Thus an indicator function $\Lambda(G, \pi)$ is a real-valued function such that $\Lambda(G, \pi) = \Lambda(G^\gamma, \pi^\gamma)$ for any $\gamma \in \Sigma_n$. (Note that $\Lambda$ does not have to be different on different isomorphism classes of graphs; if it were, then it could be used directly for isomorphism testing!) Given a node $m = (\pi; \underline{u})$ of the search tree, we define $\Lambda(m)$ to be the sequence $\Lambda(G, \pi_1), \ldots, \Lambda(G, \pi_k)$, where $\pi_0 = R(\mu)$ and $\pi_i = \pi_{i-1} \perp u_i$ for $1 \leq i \leq k$. Let $\Lambda_{\max}$ be the maximum of $\Lambda(p)$ for all terminal nodes $p$ in the search tree, under the lexicographic order. We now set

$$L_\Lambda(G) = \left\{ G^{(\sigma_\pi)} : p = (\pi; \underline{u}) \text{ is a terminal node with } \Lambda(p) = \Lambda_{\max} \right\},$$

and define $C_\Lambda(G)$ to be the $\preceq$-maximum of $L_\Lambda(G)$.

The advantage of using an indicator function is that it can also be used to prune the search tree. Suppose that a terminal node $p$ has been found. If subsequently a node $m$ is being examined where $\Lambda(m) < \Lambda(p)$, then no terminal node of $T(G, m)$ will be in $L_\Lambda(G)$ and hence $T(G, m)$ need not be examined. Note that when a nontrivial indicator function is used, the set $L_\Lambda(G)$ may be smaller than $L(G)$ and hence we may have $C_\Lambda(G) \neq C(G)$. However, $C_\Lambda(G)$ is still a canonical isomorph function, by an argument essentially identical to that of Theorem 13.

## Acknowledgements

The authors wish to thank the anonymous referee and editor Tim Chow for comments that substantially improved the presentation of this article.

## References

[1] L. Babai, Moderately exponential bound for graph isomorphism, in Fundamentals of computation theory (Szeged, 1981), Lecture Notes in Computer Science, Spinger, Berlin, 1981, v. 177, 34–50.

[2] L. Babai, D. Y. Grigoryev, and D. M. Mount, Isomorphism of graphs with bounded eigenvalue multiplicity, Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982, 310–324.

[3] H. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial $k$-trees, Journal of Algorithms 11 (1990), 631–643.

[4] D. Crnković and V. Mikulić, Block designs and strongly regular graphs constructed from the group U(3, 4), *Glas. Mat. Ser. III*, 41(61) (2006), no. 2, 189–194.

[5] A. E. Brouwer, J. H. Koolen, and R. J. Riebeek, A new distance-regular graph associated to the Mathieu group $M_{10}$, J. Alg. Comb., 8 (1998), no. 2, 153–156.

[6] I. A. Faradžev, Generation of nonisomorphic graphs with a given distribution of the degrees of vertices, Algorithmic studies in combinatorics (Russian), "Nauka", Moscow, 1978, 11–19, 185.

[7] F. Fiedler, M. H. Klin, and M. Muzychuk, Small vertex-transitive directed strongly regular graphs, *Disc. Math.*, 255 (2002), no. 1–3, 87–115.

[8] I. S. Filotti and J. N. Mayer, A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, 1980, 236–243.

[9] M. Fürer, A counterexample in graph isomorphism testing, Tech. Rep. CS-87-36, Department of Computer Science, The Pennsylvania State University, University Park, Penna., 1987.

[10] Michael R. Garey and David S. Johnson, Computers and Intractability: a guide to the theory of NP-completeness, W. H. Freeman and Co., San Francisco, CA, 1979.

[11] N. Giansiracusa, Determining graph isomorphism via canonical labeling, manuscript.

[12] J. Köbler, U. Schöning, and J. Torán, The graph isomorphism problem: its structural complexity. Progress in Theoretical Complexity, Birkhäuser Boston, Boston, MA, 1993.

[13] E. M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and System Sciences, 25 (1982), 42–65.
[14] Rudolf Mathon, A note on the graph isomorphism counting problem, Inform. Process. Lett. 8 (1979), no. 3, 131–132
[15] B. D. McKay, Practical graph isomorphism, Congr. Numer., 30 (1981), 45–87.
[16] B. D. McKay, Isomorph-free exhaustive generation, J. Algorithms, 26 (1998), no. 2, 306–324.
[17] B. D. McKay, `nauty` Software Program, Version 2.2, `http://cs.anu.edu.au/~bdm/nauty/`.
[18] B. D. McKay, and E. Rogoyski, Latin squares of order 10, Electron. J. Combin., 2 (1995), Note 3, 4 pp.
[19] G. Miller, Isomorphism testing for graphs of bounded genus, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, 1980, 225–235.
[20] T. Miyazaki, The complexity of McKay's canonical labeling algorithm. in: Groups and computation, II (New Brunswick, NJ, 1995), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 28, Amer. Math. Soc., Providence, RI, 1997, 239–256.
[21] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smirglio, Orbital branching, IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization, to appear, 2007.
[22] R. C. Read, Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations, Algorithmic aspects of combinatorics (Conf., Vancouver Island, B.C., 1976), Ann. Discrete Math., 2 (1978), 107–120.
[23] SAGE Mathematics Software, Version 3.0, `http://www.sagemath.org/`.
[24] L. Soicher, Is there a McLaughlin geometry?, J. Algebra, 300 (2006), no. 1, 248–255.
[25] L. G. Valiant, The complexity of computing the permanent, Theoret. Comput. Sci., 8 (1979), no. 2, 189–201.

Department of Mathematics, University of Nebraska–Lincoln, Lincoln, NE 68588-0130 USA

*E-mail address*: {`shartke2,aradcliffe1`}`@math.unl.edu`