An $O(n^3 \log n)$ Deterministic and an $O(n^3)$ Las Vegas Isomorphism Test for Trivalent Graphs

ZVI GALIL

Columbia University, New York, New York, and Tel-Aviv University, Tel Aviv, Israel

CHRISTOPH M. HOFFMANN

Purdue University, Lafayette, Indiana

EUGENE M. LUKS

University of Oregon, Eugene, Oregon

AND

CLAUS P. SCHNORR AND ANDREAS WEBER

University of Frankfurt, Frankfurt, West Germany

Abstract. This paper describes an $O(n^3 \log n)$ deterministic algorithm and an $O(n^3)$ Las Vegas algorithm for testing whether two given trivalent graphs on n vertices are isomorphic. In fact, the algorithms construct the set of all isomorphisms between two such graphs, presenting, in particular, generators for the group of all automorphisms of a trivalent graph. The algorithms are based upon the original polynomial-time solution to these problems by Luks but they introduce numerous speedups. These include improved permutation-group algorithms that exploit the structure of the underlying 2-groups. A remarkable property of the Las Vegas algorithm is that it computes the set of all isomorphisms between two trivalent graphs for the cost of computing only those isomorphisms that map a specified edge to a specified edge.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; *permutations and combinations*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Graph isomorphism, permutation groups, probabilistic algorithms, structure tree, time complexity, trivalent graphs, 2-groups

---

A preliminary announcement of this research appeared in *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 1982, pp. 118–125.

The research of Z. Galil was supported in part by National Science Foundation grants MCS 83-03139 and DCR 85-11713. The research of E. M. Luks was supported in part by National Science Foundation grants MCS 84-03745 and DCR 86-09491. The work of C. P. Schnorr was sponsored by the Technion, Haifa, during Summer 1981.

Authors' present addresses: Z. Galil, Computer Science Department, Columbia University, New York, NY 10027; C. M. Hoffmann, Computer Science Department, Purdue University, Lafayette, IN 47907; E. M. Luks, Department of Computer and Information Science, College of Arts and Sciences, University of Oregon, Eugene, OR 97403-1202; C. P. Schnorr, Mathematics Department and Computer Science Department, University of Frankfurt, Frankfurt, West Germany; A. Weber, Computer Science Department, University of Frankfurt, Frankfurt, West Germany.

1. *Introduction and Preliminaries*

It was only recently realized that isomorphism of trivalent graphs is decidable in polynomial time. A promising approach to graph isomorphism via permutation groups, introduced by Babai [2], stimulated work of Furst, Hopcroft, and Luks on permutation-group algorithms in general [5] and specialized algorithms for 2-groups [4]. The motivation for the latter was the natural occurrence of 2-groups in the automorphism groups of trivalent graphs (demonstrated by Tutte [13] and exploited for isomorphism-testing by both Babai [2] and Miller [10]). Luks went on to show that trivalent-graph-isomorphism is in **P** by reducing it to a color-automorphism problem for 2-groups and presenting a polynomial-time solution for the latter [8]. In fact, the technique was extended to graphs of bounded valence. In this paper, we focus on the trivalent case.

The presentation of the isomorphism test in [8] and [9] courted simplicity rather than efficiency, seeking to make the polynomial-time discovery transparent. Thus, a naive implementation seems to demand $O(n^{10})$ steps, although the initial announcement claimed an $O(n^6)$ speedup. Our new algorithms go well beyond those bounds. Deeply exploiting the underlying structures, we offer a more efficient reduction to 2-group problems as well as improved methods for computing with 2-groups.

We review Luks' algorithm in Section 2. An efficient method for dealing with 2-groups is described in Section 3. In Section 4, a simple graph-theoretic trick is employed to limit the permutation-domain blowup that occurs in passing to a color-automorphism problem. It is shown, in Section 5, that the imprimitivity blocks, which guide the basic divide-and-conquer procedure, are computable in time $O(n^2)$. A more efficient color-automorphism algorithm, exploiting the fact that one color actually predominates, is offered in Section 6; this is a simplified version of the Schnorr–Weber algorithm [11]. In Section 7, these ideas are pulled together in an $O(n^3)$ algorithm for computing generators for $\mathrm{Aut}_e(X)$, the group of automorphisms of the connected trivalent graph, X, stabilizing a specified edge e (n is the number of vertices in X). A similar procedure can be applied to determine $\mathrm{Iso}_{e_1,e_2}(X^1, X^2)$, the set of isomorphisms from X^1 to X^2 (connected trivalent graphs) that map edge e_1 to edge e_2 (Section 8). However, if the machinery has already been developed for $\mathrm{Aut}_e(X^1)$, specifically the binary tree of imprimitivity blocks together with the subgroups encountered while visiting each node, there are some shortcuts in determining the isomorphism set. In fact, with the blocks and groups precomputed and with a more economical representation of the permutations (introduced in Section 9), we demonstrate an $O(n^2 \log n)$ algorithm for $\mathrm{Iso}_{e_1,e_2}(X^1, X^2)$ (Section 10). This yields an $O(n^3 \log n)$ algorithm for testing isomorphism of trivalent graphs. In Section 11, we show how to save a $\log n$ factor by allowing coin flipping. The result is an $O(n^3)$ probabilistic algorithm that does not make errors (Las Vegas algorithm). We conclude with comments on open problems.

For a graph X, $V(X)$ denotes the set of vertices, $E(X)$ the set of edges. A graph is *trivalent* if there are at most three edges incident with any vertex.

We denote by $\mathrm{Sym}(A)$ the group of all permutations of the set A. The identity permutation and the trivial group it comprises are both denoted by **1**. Suppose G is a group acting on a set A. A subset $B \subseteq A$ is said to be *G-stable* (or *stabilized by* G) if $\sigma(B) = B$ for all $\sigma \in G$; the induced subgroup of $\mathrm{Sym}(B)$ is denoted by G^B. If G acts transitively on A (i.e., A is a single orbit), a proper subset B of A is said to be a *G-block* when, for all $\sigma, \tau \in G$, $\sigma(B) = \tau(B)$ or $\sigma(B) \cap \tau(B) = \emptyset$. We say that

G acts *primitively on* A if $|A| > 1$ and there are no G-blocks in A of size > 1. If B is a G-block in A, we call the collection $\{\sigma(B) \mid \sigma \in G\}$ a *G-block system in A*; the block system is said to be *minimal* if G acts primitively on the set of blocks. We recall that, if G is a 2-group, there are precisely two blocks in any minimal G-block system (equivalently, if a 2-group acts primitively on A, then $|A| = 2$ [6, p. 66]). Let $\pi: G \to \tilde{G}$ be a group homomorphism. An element $\sigma \in G$ is called a *lifting* of $\tilde{\sigma} \in \tilde{G}$ if $\pi(\sigma) = \tilde{\sigma}$. The kernel and image of π are denoted by $\mathrm{Ker}(\pi)$ and $\mathrm{Im}(\pi)$, respectively. If H is a subgroup of G, denoted $H \leq G$, then $[G:H]$ denotes its index in G. If $\Phi \subseteq G$, $\langle \Phi \rangle$ denotes the subgroup generated by Φ.

All run times refer to the RAM with unit cost measure [1].

2. *Luks' Algorithm*

We sketch Luks' algorithm [8, 9] for trivalent-graph-isomorphism.

It suffices, both for this original polynomial-time result and for the improved algorithms to be presented in this paper, to deal with connected trivalent graphs. By comparing pairs of connected components in the nonconnected case, the same time bounds easily extend to general trivalent graphs.

To test isomorphism of connected trivalent graphs X^1 and X^2 with n vertices and $O(n)$ edges, we answer $O(n)$ questions of the following form: Given $e_1 \in E(X^1)$ and $e_2 \in E(X^2)$, is there an isomorphism from X^1 to X^2 that maps e_1 to e_2? (Fix e_1 and try all e_2.) This question is reduced to constructing a generating set for $\mathrm{Aut}_e(X)$, the group of automorphisms of a connected trivalent graph X that fix a specified edge, e. In the reduction, the graph X is constructed, as in Figure 1, from the disjoint union of X^1 and X^2 by inserting new vertices in the "middle" of e_1, e_2, respectively, and joining these with a new edge e. The key observation is that the answer to the above question is yes iff one of the generators of $\mathrm{Aut}_e(X)$ switches the endpoints e. This reduction to the study of $\mathrm{Aut}_e(X)$ was observed in [4].

We assume, henceforth, that X is an n-vertex, connected, trivalent graph, e a designated edge of X, and we seek to compute $\mathrm{Aut}_e(X)$. The motivation for this course is twofold. Most essential is Tutte's observation [13] that $\mathrm{Aut}_e(X)$ is a 2-group. (In fact, Babai and Lovász have shown that any 2-group can arise in this fashion [3].) The other useful feature is that there is a natural sequence of "approximations" to $\mathrm{Aut}_e(X)$. For this, we let X_r, $r = 1, \ldots, n-1$, be the subgraph of X comprised of all vertices and edges on paths of length $\leq r$ through e (so X_1 is e and X_{n-1} is X). There are natural homomorphisms

$$\pi_r: \mathrm{Aut}_e(X_{r+1}) \to \mathrm{Aut}_e(X_r),$$

in which $\pi_r(\sigma)$ is the restriction of σ to X_r. In the rth stage, $r = 1, 2, \ldots$, we construct a generating set for $\mathrm{Aut}_e(X_{r+1})$ given one for $\mathrm{Aut}_e(X_r)$. For this, we solve two problems:

Problem I. Find a set of generators R for $\mathrm{Ker}(\pi_r)$.

Problem II. Find a set of generators S for $\mathrm{Im}(\pi_r)$ and a lifting S' of S to $\mathrm{Aut}_e(X_{r+1})$.

Then $R \cup S'$ generates $\mathrm{Aut}_e(X_{r+1})$. It will be clear that Problem I and the lifting of S are easy, their contribution to the time-bound insignificant.

Set $V_r = V(X_r) \backslash V(X_{r-1})$. For $v \in V_{r+1}$, we define the *neighbor set* of v to be $\{w \in V_r \mid \{v, w\} \in E(X)\}$. Neighbor sets have cardinality 1, 2, or 3. A pair u, v of

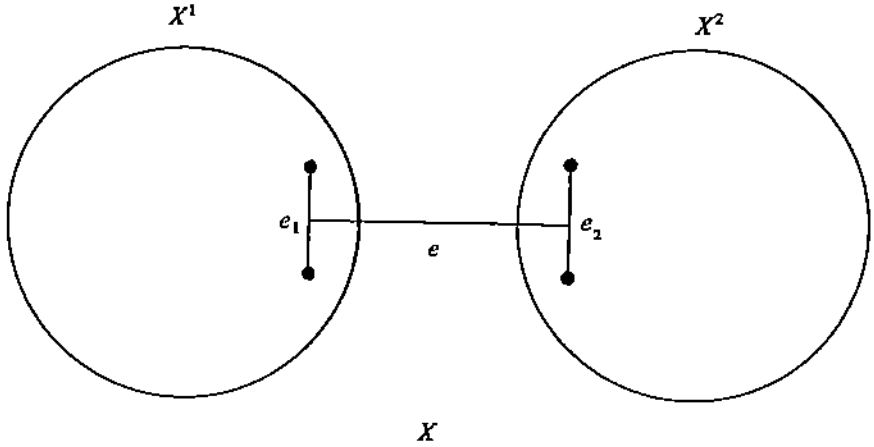

FIG. 1. The construction of X from $(X^1, e_1), (X^2, e_2)$.

vertices in V_{r+1} will be called *twins* if they have the same neighbor set. There cannot be *three* distinct vertices with a common neighbor set.

Since a permutation in $\mathrm{Ker}(\pi_r)$ fixes neighbor sets of all $v \in V_{r+1}$, its only nontrivial action can involve switching twins. For each pair, u, v of twins in V_{r+1}, let $\alpha_{\{u,v\}} \in \mathrm{Sym}(V(X_{r+1}))$ be the permutation that switches u and v while it fixes all other points. Problem I is solved by taking $\{\alpha_{\{u,v\}} \mid u, v \text{ are twins}\}$ for R.

We also derive Tutte's observation about the group structures. Note that $\mathrm{Ker}(\pi_r)$ is a 2-group, indeed, its elements have order 1 or 2. Clearly, $|\mathrm{Aut}_e(X_1)| = 2$. Suppose that $\mathrm{Aut}_e(X_r)$ is a 2-group. Then so is its subgroup $\mathrm{Im}(\pi_r)$. Since $|\mathrm{Aut}_e(X_{r+1})| = |\mathrm{Ker}(\pi_r)| \cdot |\mathrm{Im}(\pi_r)|$, it follows that $\mathrm{Aut}_e(X_{r+1})$ is a 2-group. By induction, $\mathrm{Aut}_e(X_r)$ is a 2-group for all r.

To find S, we note that $\sigma \in \mathrm{Aut}_e(X_r)$ is in $\mathrm{Im}(\pi_r)$ iff it stabilizes each of the following three collections:

(1) the collection of edges (considered as unordered pairs of vertices) connecting vertices in V_r,
(2) the collection of subsets of V_r that are neighbor sets of exactly one vertex in V_{r+1},
(3) the collection of subsets of V_r that are neighbor sets of exactly two vertices (twins) in V_{r+1}.

For any σ stabilizing (2) and (3) is liftable to $\mathrm{Sym}(V(X_{r+1}))$ by letting it map each $v \in V_{r+1}$ to a vertex whose neighbor set is $\sigma(v$'s neighbor set), the choice is unique for nontwin elements, whereas twins can be mapped to the corresponding twins in either order; if σ also stabilizes (1), this lifting is in $\mathrm{Aut}_e(X_{r+1})$. It is convenient to restate this liftability as follows: Let B_r denote the collection of all subsets of V_r of size 1, 2, or 3, and set $A_r = V(X_{r-1}) \cup B_r$; set $G_r = \mathrm{Aut}_e(X_r)$ $(\leq \mathrm{Sym}(V(X_r)))$ and (canonically) extend the action of G_r to B_r; color each element of A_r with one of five colors that distinguish

(i) whether or not it is in collection (1), and
(ii) whether it is in collection (2), or collection (3), or neither.

(Only five colors are needed since collections (1) and (3) are disjoint when $r > 1$.)

Then σ is in $\mathrm{Im}(\pi_r)$ iff σ preserves colors in A_r. Hence, Problem II is an instance of

Problem III. Given (generators for) a 2-group $G \leq \mathrm{Sym}(A)$, where A is a colored set, construct (generators for) the subgroup $\{\sigma \in G \mid \sigma \text{ preserves colors in } A\}$.

We have already incorporated one time-saver in this reduction. The algorithm in [9] takes for A_r the larger collection consisting of subsets of $V(X_r)$ of size 1, 2, or 3.

With a view toward a recursive divide-and-conquer strategy, we generalize Problem III. For $a, b \in A$, we write $a \sim b$ when a and b are similarly colored. For $K \subseteq \mathrm{Sym}(A)$ and $B \subseteq A$, let

$$C_B(K) = \{\rho \in K \mid \forall b \in B\colon \rho(b) \sim b\}.$$

We apply the function C_B only to left cosets $K = \sigma G$ of a group G that stabilizes B; for this we observe that $C_B(\sigma G)$ is either the empty set or a left coset of the subgroup $C_B(G)$. We specify (input or output) a coset of a group by enumerating generators of the group together with a coset representative. Problem III is an instance $(B = A, \sigma = \mathbf{1})$ of

Problem IV. Given a coset $\sigma G \subseteq \mathrm{Sym}(A)$, where A is a colored set and G a 2-group, and a G-stable subset, B, of A. Construct $C_B(\sigma G)$.

We present, finally, an algorithm for this problem

Algorithm 1 (*for Problem IV*).

Case 1. $(B = \{b\})$: $C_B(\sigma G) := \begin{cases} \sigma G & \text{if } \sigma(b) \sim b, \\ \varnothing & \text{otherwise;} \end{cases}$ **return**

Case 2. (G^B is intransitive): Find a nontrivial orbit B^1; set $B^2 := B \backslash B^1$; $C_B(\sigma G) := C_{B^2}C_{B^1}(\sigma G)$; **return**

Case 3. (G^B is transitive): Find a minimal G-block system $\{B^1, B^2\}$ in B; find the subgroup, H, of G that stabilizes B^1 and find some $\tau \in G \backslash H$; $C_B(\sigma G) := C_{B^2}C_{B^1}(\sigma H) \cup C_{B^2}C_{B^1}(\sigma \tau H)$; **return**

We discuss procedures for finding B^1, B^2, H, τ, in later sections. Note that Case 3 could require reformulation of the union of two nonempty cosets of $C_{B^2}C_{B^1}(H) = C_B(H)$ as a single coset (the union is known to be a coset of $C_B(G)$). Suppose that ρ_1, ρ_2 are the respective coset representatives. Then, form a generating set for $C_B(G)$ by adding $\rho_1^{-1}\rho_2$ to the generators of $C_B(H)$, and take ρ_1 as the coset representative for $C_B(\sigma G)$.

3. *Computing in 2-Groups of Permutations*

It is convenient to present 2-groups in a manner that facilitates several key computations. Let G be a 2-group generated by $\{\gamma_1, \ldots, \gamma_k\}$. Then the sequence $(\gamma_1, \ldots, \gamma_k)$ will be called a *smooth generating sequence* (SGS) for G if $[G_{(i)}\colon G_{(i-1)}] \leq 2$, for $i = 1, \ldots, k$, where $G_{(i)} = \langle \gamma_1, \ldots, \gamma_i \rangle$ (in particular, $G_{(0)} = \mathbf{1}$).

For us, the salient feature of SGSs is that, given an SGS, $(\gamma_1, \ldots, \gamma_k)$, for $G \leq \mathrm{Sym}(B)$, one can construct an SGS for a subgroup H of index 2 as well as an element $\tau \in G \backslash H$ using the results of the k membership tests "$\gamma_i \in H$?" For this:

Let

$$j = \min\{i \mid \gamma_i \notin H\}$$

and assign

$$\tau := \gamma_j;$$
$$\beta_i := \begin{cases} \gamma_i & \text{if } \gamma_i \in H, \\ \tau^{-1}\gamma_i & \text{if } \gamma_i \notin H, \end{cases} \quad \text{for } i = 1, \ldots, k.$$

LEMMA 1. *With $\beta_1, \ldots, \beta_k$ constructed as above*

(1) *$(\beta_1, \ldots, \beta_k)$ is an SGS for H.*
(2) *The time to compute this sequence is $O(k \mid B \mid)$, assuming that a membership test requires time $O(\mid B \mid)$.*

PROOF. The timing is clear. For (1), note first that $\beta_i \in H$ since $\gamma_i \notin H$ iff $\tau^{-1}\gamma_i \in H$. Let $H_{(i)} = \langle \beta_1, \ldots, \beta_i \rangle$ for $i = 0, \ldots, k$. Since β_i is either γ_i or $\gamma_j^{-1}\gamma_i$, the latter possible only for $i \geq j$, it is immediate that $\beta_i \in G_{(i)}$, and so $H_{(i)} \leq G_{(i)}$ for all i. Then, for $i > j$, $\gamma_i \notin G_{(i-1)}$ implies $\beta_i \notin H_{(i-1)}$. So, for all $i \neq j$, $[H_{(i)}: H_{(i-1)}] \geq [G_{(i)}: G_{(i-1)}]$. Observing that $[G_{(j)}: G_{(j-1)}] = 2$, we have

$$\prod_{i=1}^{k} [G_{(i)}: G_{(i-1)}] = \mid G \mid = 2 \mid H \mid \geq 2 \mid H_{(k)} \mid = 2 \prod_{i=1}^{k} [H_{(i)}: H_{(i-1)}] \geq \prod_{i=1}^{k} [G_{(i)}: G_{(i-1)}].$$

Thus equality holds throughout. We conclude that $[H_{(i)}: H_{(i-1)}] \leq 2$ for all i, and $H = H_{(k)}$. □

Observe that the length of the SGS produced for H can be reduced to $k - 1$ by omitting $\beta_j = \tau^{-1}\gamma_j = \mathbf{1}$. That will be done in applications.

Our computations with 2-groups of permutations require the use of induced actions on other sets, thus involving a homomorphic image of the current group. We next note that presentations via smooth generating sequences are preserved:

LEMMA 2. *Let $\pi: G \to \bar{G}$ be a homomorphism. If $(\gamma_1, \ldots, \gamma_k)$ is an SGS for G, then $(\pi(\gamma_1), \ldots, \pi(\gamma_k))$ is an SGS for $Im(\pi)$.*

PROOF. The image of a generating set of $G_{(i)}$ generates $\pi(G_{(i)})$ and $[\pi(G_{(i)}): \pi(G_{(i-1)})] \leq [G_{(i)}: G_{(i-1)}]$. □

The following lemma permits the merging of SGS solutions to Problems I and II.

LEMMA 3. *Let $\pi: G \to \bar{G}$ be a homomorphism. Suppose $(\beta_1, \ldots, \beta_s)$ is an SGS for $Ker(\pi)$ and $(\alpha_1, \ldots, \alpha_t)$ is an SGS for $Im(\pi)$. For $i = 1, \ldots, k$, let α_i' be any lifting of α_i to G. Then $(\beta_1, \ldots, \beta_s, \alpha_1', \ldots, \alpha_t')$ is an SGS for G.*

PROOF. Define γ_i so that $(\gamma_1, \ldots, \gamma_{s+t}) = (\beta_1, \ldots, \beta_s, \alpha_1', \ldots, \alpha_t')$. It is clear that $G = \langle \gamma_1, \ldots, \gamma_{s+t} \rangle$. For $i \leq s$, $G_{(i)} = \mathrm{Ker}(\pi)_{(i)}$ and, for $i > s$, $\pi(G_{(i)}) = \pi(G)_{(i-s)}$ (where the subscripted subgroups are defined relative to the appropriate generating sequence). Then

$$[G_{(i)}: G_{(i-1)}] = \begin{cases} [\mathrm{Ker}(\pi)_{(i)}: \mathrm{Ker}(\pi)_{(i-1)}] & \text{if } i \leq s, \\ [\pi(G)_{(i-s)}: \pi(G)_{(i-s-1)}] & \text{if } i > s. \end{cases}$$

In any case, $[G_{(i)}: G_{(i-1)}] \leq 2$. □

Remark 1. Any 2-group G has an SGS $(\gamma_1, \ldots, \gamma_k)$ of minimal length, that is, with $k = \log_2(\mid G \mid)$. We do not, however, insist on that restriction in all discussions involving SGSs. The reason is that we apply some of the results (those of Section 5, in particular) to induced, possibly unfaithful, actions of a group; indeed,

Lemma 2 is offered with that in mind. Nevertheless, in our applications of a revised Algorithm 1, there is an underlying embedding of the group in $\mathrm{Sym}(V(X_r))$. With this full group in view, the SGS will always have minimal length. Thus the length, k, of any SGS will never exceed $n/2$ where $n = |V(X)|$ (recall that $|\mathrm{Ker}(\pi_r)| = 2^s$, where $s =$ the number of twins in $V_{r+1} \leq |V_{r+1}|/2$; so, an induction on r shows $\log_2(|\mathrm{Aut}_e(X_r)|) \leq |V(X_r)|/2$ for all r).

4. *Eliminating the Triples in B_r*

The size of B_r is $O(m_r^3)$ where $m_r = |V_r|$. A graph-theoretic trick is used to eliminate the triples in B_r, reducing its size to $O(m_r^2)$.

Recall that triples are incorporated because the neighbor set of a vertex $v \in V_{r+1}$ could have cardinality 3. This situation can be avoided by replacing each such v by a triangle with vertices at "level" $r+1$ (see Figure 2) and having *labeled* (with +) edges. The result is an edge-labeled graph denoted by $\tilde{X}$ with neighbor sets of size ≤ 2.

In considering $\mathrm{Aut}_e(\tilde{X})$, it is presumed that automorphisms map labeled edges to labeled edges. With this convention, the homomorphism h: $\tilde{X} \to X$, contracting each labeled triangle to a vertex, induces an isomorphism Ψ_h: $\mathrm{Aut}_e(\tilde{X}) \cong \mathrm{Aut}_e(X)$. The computation of $\mathrm{Aut}_e(\tilde{X})$ follows that of $\mathrm{Aut}_e(X)$ except: B_r need only include the subsets of (the modified) V_r of size 1 or 2; collection (1) is split into

(1a) the collection of unlabeled edges connecting vertices in V_r,
(1b) the collection of labeled edges connecting vertices in V_r;

and an additional color is allowed for an element of A_r to distinguish:

(i′) whether it is in collection (1a), or collection (1b), or neither.

Note that only one color is added since collection (1b) is disjoint from collections (2) and (3). Thus, replacing X by $\tilde{X}$, we may assume B_r is free of triples.

Also, we reformulate B_r as the set of *ordered* pairs in V_r. For this, observe first that the coloring of B_r induces a coloring of $V_r \times V_r$ in which (v, v) has the color of v, while both (u, v) and (v, u) inherit the color of $\{u, v\}$. With this color assignment, the reassignment

$$B_r := V_r \times V_r$$

retains the identification of $\mathrm{Im}(\pi_r)$ with the color-preserving subgroup, $C_{B_r}(\mathrm{Aut}_e(X_r))$. Henceforth, we take this definition of B_r.

5. *Precomputing the Blocks*

Algorithm 1 evokes recursive calls for $C_{\tilde{B}}(\tilde{\sigma}\tilde{G})$ with $\tilde{B} \subseteq B$, $\tilde{G} \leq G$. The work can be reorganized so as to limit the number of distinct blocks, $\tilde{B}$, visited. These blocks form a tree that is precomputed and guides the recursion.

Let G be a 2-group acting on B. We call a binary tree T a *structure tree for B with respect to G* (notation $T = \mathrm{Tree}(B, G)$), if

(1) the set of leaves of T is B,
(2) the action of any $\sigma \in G$ on B can be lifted to an automorphism of T.

The nodes in a structure tree may be identified with and labeled by their set of descendent leaves. Each node is itself the root of a structure tree for the action of any subgroup that stabilizes the corresponding subset. Thus, in Algorithm 1, the

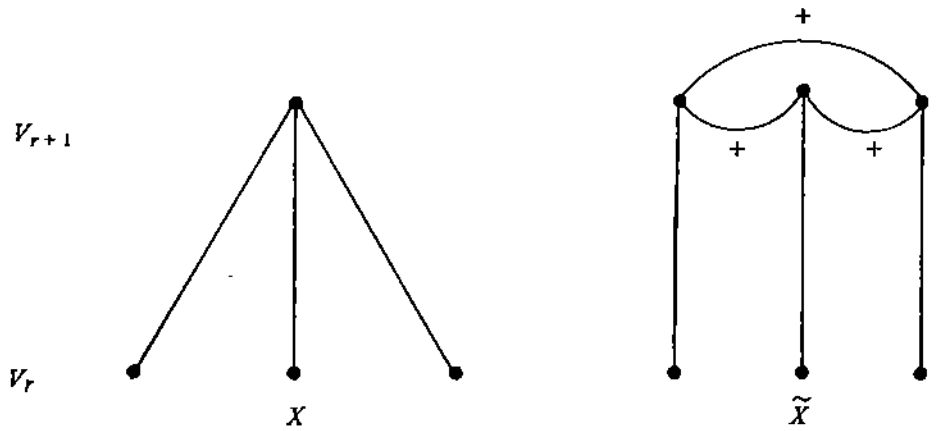

FIG. 2. Replacing the triple neighbor sets.

subsets B^1 and B^2 correspond to the left and right sons, B_L and B_R, of node B. (There will be no confusion in context with the notation B_r for the rth stage target set.) With $G = \langle \gamma_1, \ldots, \gamma_k \rangle$, Cases 2 and 3 of the Algorithm 1 can be rephrased:

(*New*) *Case* 2. $(\forall j \leq k\colon \gamma_j(B_L) = B_L)$: $C_B(\sigma G) := C_{B_R} C_{B_L}(\sigma G)$; **return**

(*New*) *Case* 3. $(\exists j \leq k\colon \gamma_j(B_L) = B_R)$: find the subgroup H of G that stabilizes B_L and find some $\tau \in G \backslash H$; $C_B(\sigma G) := C_{B_R} C_{B_L}(\sigma H) \cup C_{B_R} C_{B_L}(\sigma \tau H)$; **return**

Again, one uses the subtrees rooted at B_L, B_R in the evaluation of C_{B_L}, C_{B_R}. We precompute the entire structure tree for the *initial* (B, G) as follows:

Algorithm 2 (*construction of* $T = Tree(B, G)$).

1. let the root of T be B;
2. **if** $|B| = 1$ **then return** (* B is a leaf *)
3. find the orbits of G in B; **if** G is transitive on B **then goto** 4; partition B into two nontrivial G-stable subsets (* i.e., unions of orbits *) B_L, B_R; recursively compute Tree(B_L, G), Tree(B_R, G); construct T to be the union, joined to the new root B, of Tree(B_L, G) and Tree(B_R, G); **return**
4. find a minimal block system $\{B_L, B_R\}$ for G on B; find the subgroup H of G that stabilizes B_L and find some $\tau \in G \backslash H$; recursively compute Tree(B_L, H); construct T to be the union, joined to the new root B, of Tree(B_L, H) and τ(Tree(B_L, H)) (* i.e., the tree, rooted at B_R, formed by the τ-images of the nodes of Tree(B_L, H) *); **return**

Remark 2. One can modify the construction of T so as to produce a complete binary tree. For this, add points to the initial B so that $|B| = 2^s$ for some s, letting the elements of G act trivially on the new points. Then, one need only take care, in the nontransitive case, to partition B into two equal parts (this is always possible since the cardinality of each orbit is a power of 2).

We establish bounds on the set decompositions in Algorithm 2. In the following, we let $\Psi(x, y)$ denote the time bound for union-find with x operations on y elements [1]. It is convenient to postpone the application of a good upper bound for $\Psi(x, y)$ in order to capitalize on the following direct consequence of the definition

$$(*) \qquad \Psi(x_1, y_1) + \Psi(x_2, y_2) \leq \Psi(x_1 + x_2, y_1 + y_2).$$

LEMMA 4. *Given an SGS* $(\gamma_1, \ldots, \gamma_k)$ *for* $G \leq Sym(B)$, $|B| = m$,

(1) *the orbits of* G *in* B *can be computed in time* $O(km)$,
(2) *if* G^B *is transitive, a minimal block system* $\{B_L, B_R\}$ *for* G *on* B *can be computed in time* $O(\Psi(2km, 2m))$.

PROOF

(1) Finding the orbits is equivalent to finding the connected components in the undirected graph with vertex set B and edge set $\{\{b, \gamma_i(b)\} \mid b \in B, i = 1, \ldots, k\}$.

(2) For $b_1, b_2 \in B$, we write $b_1 \approx b_2$ if there is a minimal block system $\{B^1, B^2\}$ in B with $b_1, b_2 \in B^1$. To test this condition, we determine the *unique* smallest G-block containing both b_1 and b_2; the result is a proper subset iff $b_1 \approx b_2$. We point out that the block system including this smallest block is the equivalence class decomposition for the finest G-invariant equivalence relation, R, in which b_1 R b_2 (i.e., the equivalence classes are forced by: b_1 R b_2, and x R y implies $\gamma_i(x)$ R $\gamma_i(y)$ for $i \le k$). To construct R, we use the algorithm in [1, p. 144] that tests equivalence of deterministic finite automata. (It assumes that the initial states are equivalent and computes the equivalence classes implied by the two transition functions.) This can be carried out in time $O(\Psi(km, m))$. To find a pair $b_1 \approx b_2$ it suffices to take any three elements in B and make the three pairwise tests; at least one test must succeed since some two of the elements are together in any minimal block system. If the successful test reveals exactly two blocks, we are done. Otherwise, we have a partition $\mathbf{B} = \{B^1, \ldots, B^s\}$ of B into s blocks of equal size m/s. We continue, recursively, with the action of G on $\mathbf{B}$. Thus, the time-bound for (2), $t(m, k)$, satisfies

$$t(m, k) \le O(\Psi(km, m)) + t\left(\frac{m}{2}, k\right).$$

The result follows by (*). □

This leads to a bound for the construction of Tree(B, G).

LEMMA 5. *Given an SGS* $(\gamma_1, \ldots, \gamma_k)$ *for* $G \le Sym(B)$, $|B| = m$, *a structure tree Tree*(B, G) *can be computed in time* $O(\Psi(4km, 4m))$.

PROOF. Let $t(m, k)$ be the time bound for constructing Tree(B, G) given B of size m and an SGS for G of length k. By Lemma 4, the G-orbit decomposition, $B = \bigcup_i D_i$, can be found in time $O(km)$. To form Tree(B, G), we need the subtrees Tree(D_i, G) (see step 3 of Algorithm 2). In constructing each of these, we restrict the SGS to its image in G^B (by Lemma 2, it remains an SGS). Let $d_i = |D_i|$, so $\sum_i d_i = m$. For each i with $d_i > 1$, we construct a minimal G-block system $\{D_i^1, D_i^2\}$ in D_i within time $O(\Psi(2kd_i, 2d_i))$ using Lemma 4. We then find the subgroup H_i of G^{D_i} that stabilizes D_i^1 and we find some $\tau_i \in G^{D_i} \backslash H_i$. By Lemma 1, this takes time $O(kd_i)$. We construct Tree(D_i^1, H_i) in time $t(d_i/2, k - 1)$ and then τ(Tree(D_i^1, H_i)) in time $O(d_i)$, joining these to form Tree(D_i, G) (see step 4 of Algorithm 2). We have

$$\begin{aligned} t(m, k) &= O(km) + \sum_i t(d_i, k) \\ &\le O(km) + \sum_i c\Psi(2kd_i, 2d_i) + \sum_i t\left(\frac{d_i}{2}, k - 1\right) \qquad (c \text{ a constant}) \\ &= O(\Psi(2km, 2m)) + \sum_i t\left(\frac{d_i}{2}, k - 1\right). \end{aligned}$$

Hence, by (*) and induction, $t(m, k) \le O(\Psi(4km, 4m))$. □

In our reduction to $n - 2$ stages of color-automorphism problems, the groups that arise are the $G_r = \mathrm{Aut}_e(X_r) \le \mathrm{Sym}(A_r)$, and the colored sets $A_r = V(X_{r-1}) \cup B_r$. But, since $V(X_{r-1})$ is both G-stable and homogeneously colored, it

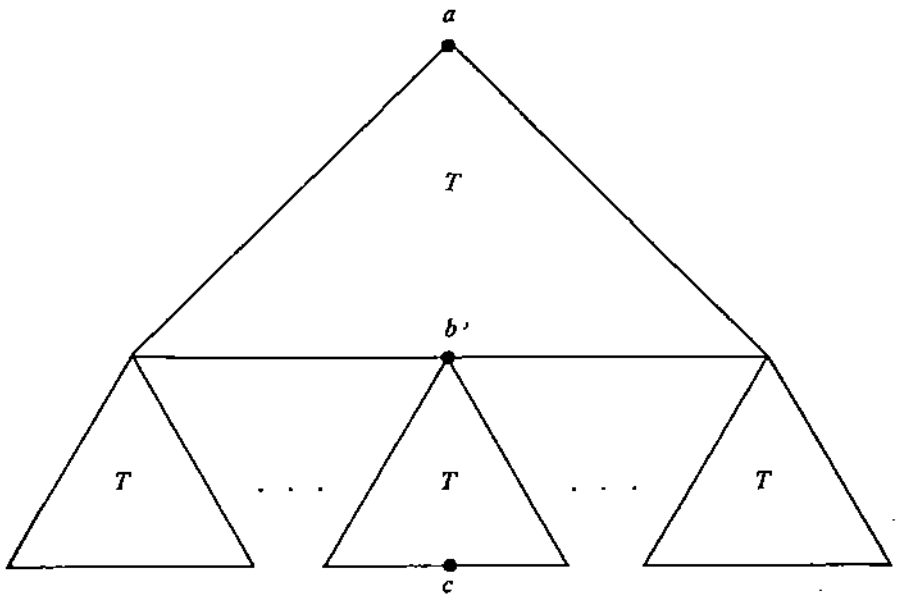

FIG. 3. Tree(B_r, G_r) with three typical nodes $a = B_r$, $b = \{v\} \times V_r$, $c = \{(v, w)\}$.

suffices to stabilize colors in B_r. Thus, we need the structure trees, Tree(B_r, G_r), for each stage $r = 1, \ldots, n - 2$. Recall that $B_r = V_r \times V_r$, where $V_r = V(X_r) \backslash V(X_{r-1})$, and $m_r := |V_r|$.

LEMMA 6. *Given an SGS* $(\gamma_1, \ldots, \gamma_k)$ *for* G_r, *a structure tree Tree*(B_r, G_r) *can be constructed in time* $O(\Psi(4km_r, 4m_r) + m_r^2)$.

PROOF. A structure tree Tree($V_r \times V_r$, G_r) can be constructed, as in Figure 3, from $m_r + 1$ copies of Tree(V_r, G_r). (There is one upper and m_r lower copies; in the upper copy node B is relabeled $B \times V_r$; for each $v \in V_r$, there is a lower subtree in which node B is relabeled $\{v\} \times B$.) By Lemma 5, Tree(V_r, G_r) can be formed in time $O(\Psi(4km_r, 4m_r))$ and, from that, it takes $O(m_r^2)$ steps to construct Tree($V_r \times V_r$, G_r). □

To determine the effect of the last estimate over all stages, observe that k and $\sum_{r=1}^{n-2} m_r$ are bounded above by n. We have $\sum_r m_r^2 \leq n^2$ and, by (*), $\sum_r \Psi(4km_r, 4m_r) \leq \Psi(4n^2, 4n)$. But Tarjan has shown $\Psi(x, y) = O(x\alpha(x, y))$ for $x \geq y$ (see [12] for this and for a discussion of α). Since $\alpha(4n^2, 4n) = O(1)$, we conclude

THEOREM 1. *The structure trees Tree*(B_r, G_r) *for all stages,* $r = 1, \ldots, n - 2$, *can be constructed in total time* $O(n^2)$.

6. *Early Termination, the Groups, and the Main Color*

Algorithm 1 terminates (that is, avoids deeper recursion) in Case 1 where $|B| = 1$. We indicate two other situations in which termination could be advanced. Let Q_i denote the set of elements in A with color i. If, for any i, $|B \cap Q_i| \neq |\sigma(B) \cap Q_i|$, then $C_B(\sigma G) = \emptyset$. Also, if $B \subseteq Q_i$ and $\sigma(B) \subseteq Q_i$ then $C_B(\sigma G) = \sigma G$. We expand Case 1 to include these tests:

Case 1a. $(\exists i: |B \cap Q_i| \neq |\sigma(B) \cap Q_i|)$: $C_B(\sigma G) := \emptyset$; **return**

Case 1b. $(\exists i: B \cup \sigma(B) \subseteq Q_i)$: $C_B(\sigma G) := \sigma G$; **return**

These additional termination opportunities have a significant effect on the timing. In the rest of this section, we develop some machinery for analyzing that effect and cutting some additional costs.

We point out first that the determination of the index 2 subgroup, in Case 3, may be repetitious, for the problem recurs with the same G. There are, in fact, two subgroups associated to each node $\tilde{B}$ of Tree(B, G). To describe these, let $L(\tilde{B})$, $R(\tilde{B})$ denote the set of leaves of Tree(B, G) to the left and right, respectively, of $\tilde{B}$. Let $G_{\tilde{B}}$ be the subgroup of G that stabilizes the sets $L(\tilde{B})$, $\tilde{B}$, $R(\tilde{B})$ and preserves colors in $L(\tilde{B})$. It follows that $C_{\tilde{B}}(G_{\tilde{B}})$ is the subgroup of G that stabilizes $L(\tilde{B})$, $\tilde{B}$, $R(\tilde{B})$ and preserves colors in $L(\tilde{B}) \cup \tilde{B}$. The groups $G_{\tilde{B}}$ and $C_{\tilde{B}}(G_{\tilde{B}})$ may be thought of as the *entry* and *exit* groups for $\tilde{B}$, for we have

LEMMA 7. *In Algorithm* 1, *the coset passed in each call of the form* $C_{\tilde{B}}$ *is a coset of* $G_{\tilde{B}}$ (*hence, if a nonempty answer is returned, it is a coset of* $C_{\tilde{B}}(G_{\tilde{B}})$). *If we agree that the coset* σH *is always treated first in Case* 3, *then the first* $C_{\tilde{B}}$ *call passes the group* $G_{\tilde{B}}$ *itself.*

PROOF. The first statement follows by induction on the postorder number [1, p. 54] of the nodes of Tree(B, G), the second by induction on the preorder number. □

A nonleaf $\tilde{B}$ of Tree(B, G) is called *transitive* if the entry group, $G_{\tilde{B}}$, acts transitively on the (two-element) set $\{\tilde{B}_L, \tilde{B}_R\}$ and *intransitive*, otherwise. A transitive node $\tilde{B}$ is called *color-transitive* if the exit group, $C_{\tilde{B}}(G_{\tilde{B}})$, acts transitively on $\{\tilde{B}_L, \tilde{B}_R\}$. Note that the conditions on (New) Cases 2 and 3 could be restated

Case 2. (B is intransitive).

Case 3. (B is transitive).

The following properties of entry, exit groups are immediate.

LEMMA 8. *Let* $\tilde{B}$ *be any nonleaf of Tree*(B, G), $\tilde{B}_L$, $\tilde{B}_R$ *its sons.*

(1) $[G_{\tilde{B}} : G_{\tilde{B}_L}] \leq 2$, *with equality holding iff* $\tilde{B}$ *is transitive,*
(2) $C_{\tilde{B}_L}(G_{\tilde{B}_L}) = G_{\tilde{B}_R}$,
(3) $[C_{\tilde{B}}(G_{\tilde{B}}) : C_{\tilde{B}_R}(G_{\tilde{B}_R})] \leq 2$, *with equality holding iff* $\tilde{B}$ *is color-transitive.*

The additional termination opportunities offered in Cases 1a and 1b of New Algorithm 1 have a significant effect on the timing. This is due, in part, to the fact that one of the colors tends to force Case 1b. In the color-automorphism problems arising from the computation of $\mathrm{Aut}_e(X_{r+1})$, six colors are used to distinguish membership in collections (2), (3) (Section 2) and (1a), (1b) (Section 4). These colors are not evenly distributed. One color, we'll say it is the sixth, signaling nonmembership in any of these four collections, predominates. All of $V(X_{r-1})$ has the sixth color. Furthermore, any $v \in V_r$ is the first coordinate of at most two pairs in $V_r \times V_r$ that display one of the first five colors. (If (v, u) is so colored then either $\{v, u\}$ is a new edge in X_{r+1} or there are edges from v and u to a vertex in V_{r+1}.) With this in mind, we ignore the sixth color, considering Q_6 to be *uncolored.* As a result, the set $Q = \bigcup_{i<6} Q_i$, of colored elements in $B_r = V_r \times V_r$ (equivalently, the set of colored elements in A_r) has cardinality at most $2m_r$. It suffices to guarantee color-preservation in Q, though we continue to consider $i = 6$ in Cases 1a and 1b.

We modify the structure tree $T = \mathrm{Tree}(B, G)$ to exploit the above interpretation of the coloring. A node $\tilde{B}$ of T will be called *inactive* if $\tilde{B} \cap Q = \emptyset$, and *active* otherwise. We say that the node $\tilde{B}$ is *visited* each time a call to $C_{\tilde{B}}$ does not

terminate in Cases 1a and 1b. It is clear that, with the incorporation of those cases, only active nodes of T are visited during the recursion. With this in mind, we consider the subtree $\text{Tree}_p(B, G)$, consisting of the active nodes, referring to it as the *pruned* tree. The pruned tree still guides the recursion.

Amongst the active nodes, there are some that will contribute little to the cost of the algorithm. We call an active node $\tilde{B}$ *facile* if $\tilde{B}$ is intransitive with exactly one active son, and *nonfacile* otherwise. We assume now, that in the construction of the pruned tree, we have marked facile nodes and, by switching labels, when necessary, have assured that $\tilde{B}_L$ is the active son of facile node $\tilde{B}$. To see how we save work at these nodes, suppose $\tilde{B}$ is facile and that the guard on Case 1a was not satisfied. Let $\Delta(\tilde{B})$ denote the nearest nonfacile descendent of $\tilde{B}$. Then, if $\alpha \in \tilde{\sigma} G_{\tilde{B}}$ is color-preserving on $\Delta(\tilde{B})$, it must be color-preserving on $\tilde{B}$. Hence, $C_{\tilde{B}}(\tilde{\sigma} G_{\tilde{B}}) = C_{\Delta(\tilde{B})}(\tilde{\sigma} G_{\tilde{B}})$, so that we can pass to node $\Delta(\tilde{B})$.

We incorporate the above, together with the modifications from Section 5 in:

New Algorithm 1 *(for Problem IV, given Tree(B, G) and an SGS for G).*
Case 1a. $(\exists i\colon |B \cap Q_i| \neq |\sigma(B) \cap Q_i|)$: $C_B(\sigma G) := \varnothing$; **return**
Case 1b. $(\exists i\colon B \cup \sigma(B) \subseteq Q_i)$: $C_B(\sigma G) := \sigma G$; **return**
Case 1.5 (B is facile): $C_B(\sigma G) := C_{\Delta(B)}(\sigma G)$; **return**
Case 2. (B is intransitive): $C_B(\sigma G) := C_{B_R} C_{B_L}(\sigma G)$; **return**
Case 3. (B is transitive): Find the subgroup H of G that stabilizes B_L and find some $\tau \in G \backslash H$; $C_B(\sigma G) := C_{B_R} C_{B_L}(\sigma H) \cup C_{B_R} C_{B_L}(\sigma \tau H)$; **return**

It is assumed that the guards are tested in the order shown. For the timing argument, we determine, in the following three lemmas, some bounds on the size of the pruned tree.

LEMMA 9. *Assuming $\text{Tree}(B_r, G_r)$ is constructed as a complete binary tree (see Remark 2), it has at most $O(m_r \log m_r)$ active nodes.*

PROOF. The pruned tree $\text{Tree}_p(B_r, G_r)$ has at most $2m_r$ leaves. Since the complete binary tree $\text{Tree}(G_r, B_r)$ has m_r^2 leaves, all paths within it, hence all paths within the pruned tree, have length at most $2 \log_2 m_r$. □

LEMMA 10. *There are at most $2m_r$ intransitive, nonfacile nodes in the pruned tree $\text{Tree}_p(B_r, G_r)$.*

PROOF. Each intransitive, nonfacile node has two sons in the pruned tree, which has $\leq 2m_r$ leaves. □

LEMMA 11. *There are at most $2.5n$ transitive nodes in all pruned trees, $\text{Tree}_p(B_r, G_r)$, $r = 1, 2, \ldots, n-2$.*

PROOF. At each stage of the computation of $\text{Aut}_e(X)$, the group gains some generators and loses some. (Recall SGSs always have minimal length.) In stage r, generators of the corresponding $\text{Ker}(\pi_r)$ are gained; incorporating the starting generator of $\text{Aut}_e(X_1)$, their total, over all r, is at most $n/2$. There is also a possible loss at stage r in passing from $G_r = \text{Aut}_e(X_r)$ to $\text{Im}(\pi_r)$. But, by Lemma 8 and induction on the tree height, one sees that the number of generators lost in passing from an entry group, G_B, to the exit group, $C_B(G_B)$ is the number of transitive but not color-transitive nodes in $\text{Tree}(B, G)$. Since one cannot lose more than one gains, the number of transitive but not color-transitive nodes cannot total more than $n/2$ over all stages. (A similar argument is used in [7].) On the other hand, the number of color-transitive nodes in stage r cannot exceed the number of colored leaves, so that, over all stages, the number of color-transitive nodes is bounded by $\sum_r 2m_r \leq 2n$. The result follows. □

Finally, we need a bound on the number of visits to any node. For a node $\tilde{B}$ in $\mathrm{Tree}_p(B_r, G_r)$, let $\mathrm{vis}(\tilde{B})$ be the number of times $\tilde{B}$ is visited during the recursion of New Algorithm 1. By virtue of Case 1.5 we have eliminated any calls at all to certain facile nodes (those strictly between a facile son, $\tilde{B}$, of a nonfacile node and $\Delta(\tilde{B})$). It is immediate, if a facile node $\tilde{B}$ is ever visited, that

$$\mathrm{vis}(\Delta(\tilde{B})) \leq \mathrm{vis}(\tilde{B}).$$

For the sons $\tilde{B}_L$, $\tilde{B}_R$, of an arbitrary $\tilde{B}$, we have

$$\mathrm{vis}(\tilde{B}_L),\ \mathrm{vis}(\tilde{B}_R) \leq \begin{cases} \mathrm{vis}(\tilde{B}) & \text{if } \tilde{B} \text{ is intransitive,} \\ 2\ \mathrm{vis}(\tilde{B}) & \text{if } \tilde{B} \text{ is transitive.} \end{cases}$$

For transitive nodes with "unequal color partition" there is a better bound:

$$\exists j\colon |\tilde{B}_L \cap Q_j| \neq |\tilde{B}_R \cap Q_j| \Rightarrow \mathrm{vis}(\tilde{B}_L),\ \mathrm{vis}(\tilde{B}_R) \leq \mathrm{vis}(\tilde{B}).$$

To see this, suppose some visit $C_{\tilde{B}}(\sigma G_{\tilde{B}})$ to $\tilde{B}$ engenders more than one visit to son $\tilde{B}_L$ (the same argument applies for $\tilde{B}_R$). Then, since two subsequent calls to $C_{\tilde{B}_L}$ escaped Case 1a, we have, for all j

$$|\tilde{B}_L \cap Q_j| = |\sigma(\tilde{B}_L) \cap Q_j| = |\sigma\tau(\tilde{B}_L) \cap Q_j|.$$

But $\sigma(\tilde{B}_L) \cup \sigma\tau(\tilde{B}_L) = \sigma(\tilde{B})$, and, since this started with a visit to $\tilde{B}$, $|\tilde{B} \cap Q_j| = |\sigma(\tilde{B}) \cap Q_j|$, for all j. Hence, for all j, $|\tilde{B}_L \cap Q_j|$ is half of $|\tilde{B} \cap Q_j|$, so that $|\tilde{B}_L \cap Q_j| = |\tilde{B}_R \cap Q_j|$.

These relations and an induction on the distance from the root B establish that $\mathrm{vis}(\tilde{B}) \leq 2m_r/|\tilde{B} \cap Q|$. Since a call to any node $\tilde{B}$ can only follow a visit to a parent or a unique facile ancestor, we conclude

LEMMA 12. *For any node $\tilde{B}$ of $\mathrm{Tree}_p(B_r, G_r)$ there are at most $4m_r$ calls to $C_{\tilde{B}}$ during the recursion of New Algorithm* 1.

7. *The Time Bound for Computing $\mathrm{Aut}_e(X)$*

By Theorem 1, the structure trees $\mathrm{Tree}(B_r, G_r)$, for all stages, $r = 1, 2, \ldots$, are found in time $O(n^2)$. By Lemma 9, pruning, including the construction of the function Δ, takes $O(n \log n)$ steps. We also need

LEMMA 13

(1) *The entry groups for all nodes of the structure trees and the τ's (for Case* 3 *of New Algorithm* 1*) are computed in time $O(n^3)$.*
(2) *Transitivity is tested for all nodes in time $O(n^2 \log n)$.*

PROOF

(1) We compute new groups and τ's only at transitive active nodes. By Lemma 11, there are $O(n)$ such nodes. By Lemma 1, the cost at each is $O(kn)$, where k ($<n$) is the length of the SGS for the entry group.
(2) Transitivity is tested at an active node in $O(k)$ steps and only on the first visit to that node. Using Lemma 9, the result follows. □

We now prove

THEOREM 2. *Let X be an n-vertex, connected, trivalent graph. Then $\mathrm{Aut}_e(X)$ can be computed in time $O(n^3)$.*

PROOF. We use Algorithm 2 for the blocks and New Algorithm 1 for Problem IV. By Lemma 13, we may assume that all the groups, τ's, and transitive

nodes are given. At each call $C_{\tilde{B}}$ to a node $\tilde{B}$, we execute the tests in Cases 1a and 1b and, if the node is transitive, we might have to multiply two permutations in Case 3. The cost of such a call is $O(n)$. By Lemma 12, there are at most $O(m_r) = O(n)$ calls to any node. By Lemmas 10 and 11, there are $O(n)$ nodes in all pruned trees which are either transitive or nonfacile. Thus the cost associated to all called nodes is $O(n^3)$. □

8. *The Isomorphism Test*

Having established an $O(n^3)$ algorithm for $\mathrm{Aut}_e(X)$, the reduction in Section 2 could be used to test isomorphism of n-vertex, connected, trivalent graphs X^1, X^2 in $O(n^4)$ steps. That would involve an $\mathrm{Aut}_e(X)$ computation corresponding to each $e_2 \in E(X^2)$. An examination of this process, however, discloses repetitive computation of the groups, blocks, etc. for $\mathrm{Aut}_{e_1}(X^1)$. We demonstrate an effective reorganization of the work, capitalizing on the precomputation and storage of that information.

Our goal is to compute $\mathrm{Iso}_{e_1,e_2}(X^1, X^2)$, the set of all isomorphisms from X^1 to X^2 that take edge e_1 to edge e_2. If σ is one such isomorphism, then $\mathrm{Iso}_{e_1,e_2}(X^1, X^2) = \sigma\mathrm{Aut}_{e_1}(X^1)$. With this in mind, we compute $\mathrm{Aut}_{e_1}(X^1)$, together with the blocks, groups, τ's as in earlier sections, and seek a single representative isomorphism, if one exists. Actually, our interest in the question was only to determine whether such σ exists. However, our method of approximation-by-stages requires knowledge of the entire isomorphism set at each intermediate stage anyway. Thus, the graphs are approximated by the subgraph sequences X^1_r, X^2_r, $r = 1, 2, \ldots, n-1$. We construct, inductively, $\sigma_{r+1} \in \mathrm{Iso}_{e_1,e_2}(X^1_{r+1}, X^2_{r+1})$ given $\sigma_r \in \mathrm{Iso}_{e_1,e_2}(X^1_r, X^2_r)$.

Analogous to the discussion of Section 2, this reduces to

Problem V. Given a 2-group $G \leq \mathrm{Sym}(A)$, a bijection $\sigma: A \to A'$, where A, A' are colored sets, and a G-stable subset, B, of A. Determine whether $\bar{C}_B(\sigma G)$ is nonempty, where

$$\bar{C}_B(\sigma G) = \{\rho \in {}^{\iota}\sigma G \mid \forall b \in B : \rho(b) \sim b\}$$
$$= \bar{\sigma} C_B(G) \quad \text{if} \quad \bar{\sigma} \text{ is any element of } \bar{C}_B(\sigma G),$$

and, if so, exhibit an element $\bar{\sigma}$ therein.

Note that the sets σG now consist of bijections from A to A'. For our purposes, they behave like the cosets of earlier sections.

We attack Problem V using analogues of the method for Problem IV. In fact, with suitable reinterpretation of σ, and using the induced tree structure $\sigma(\mathrm{Tree}(B, G))$ on $\sigma(B)$, New Algorithm 1 applies. However, since our goal is to tackle multiple instances with the same A, B, and G, we first compute $C_B(G)$ (using Algorithm 2 and New Algorithm 1), saving all the groups and structure tree information. In the isomorphism application, we assume that $\mathrm{Aut}_{e_1}(X^1)$ is computed and the data saved for all stages.

The prepossession of this wealth of knowledge does not, itself, constitute a speed-up. It eliminates one $O(n^3)$ bottleneck (see Lemma 13) but there is another (attributable to the costs of testing Cases 1a and 1b and multiplying permutations; see proof of Theorem 2). We still need to speed up the latter.

9. *Local Representation of the Maps*

We approach Problem V with the groups and structure tree in hand. The technique applies as well to Problem IV under the same assumption (where we then need

only to determine the nonemptiness of the solution coset, and, if nonempty, a single coset representative). In fact, it may be useful to keep that familiar context in mind in the discussion leading to Algorithm 3 (for that, just suppose $A = A'$).

As indicated, the cost per node is now dominated by the time to execute the tests in Cases 1a and 1b and the time to compose σ and τ in Case 3. This time will be reduced by representing the σ's relative to the node (block) under consideration.

For any $\sigma: A \to A'$ and node $\tilde{B}$ in Tree(B, G), we define the *local representative of σ at $\tilde{B}$*, notation $\sigma[\tilde{B}]$, by

$$\sigma[\tilde{B}] := \{(b, \sigma(b)) \mid b \in \tilde{B}, \sigma(b) \in Q\},$$

where $Q = \cup_{i<6} Q_i$ is now the set of colored elements (i.e., with "main" color Q_6 excluded) in $A \cup A'$. We assume also that each node $\tilde{B}$ contains the list of colored elements in it. Let $q(\tilde{B}) = |\tilde{B} \cap Q|$.

It is our intention to pass only the local representatives to the modified procedure. We consider the effect on the local representative of essential manipulations on σ.

Given $\sigma[B]$ and $B \cap Q$, we can perform the Cases 1a and 1b tests

$$(\exists i: |B \cap Q_i| \neq |\sigma(B) \cap Q_i|),$$
$$(\exists i: B \cup \sigma(B) \subseteq Q_i),$$

in time $O(q(B))$.

In Case 3, given $\tau \in \mathrm{Sym}(A)$ and $\sigma[B]$, we want to compute $\sigma\tau[B]$. This, too, can be done in $O(q(B))$ steps, for

$$\sigma\tau[B] = \{(\tau^{-1}(b), c) \mid (b, c) \in \sigma[B]\},$$

and, having passed Case 1a, $q(B) = |\sigma[B]|$. We refer to this operation as a *B-product*.

The first problem arises in Cases 2 and 3 with the progression from node B_L to node B_R. The output of the $\bar{C}_{B_L}$ call will contain only a local representative, $\alpha[B_L]$, for $\alpha \in \bar{C}_{B_L}(\sigma G_{B_L})$. However, to determine $\bar{C}_{B_R}\bar{C}_{B_L}(\sigma G_B) = \bar{C}_{B_R}(\alpha C_{B_L}(G_{B_L}))$, we need $\alpha[B_R] = \alpha[B] \backslash \alpha[B_L]$. We compute this by repeating the computation of $\bar{C}_{B_L}(\sigma G_{B_L})$ (to be precise, a sequence of $\tilde{B}$-products with known τ's) with $\sigma[B_R]$ replacing $\sigma[B_L]$. We refer to the resulting local representative at B, $\alpha[B]$, as the *expansion of* $\alpha[B_L]$ to B. The point to make is that the repeated computation needs at most one visit to each node in the pruned subtree rooted at B_L. In the first pass, multiple visits were forced because it was not known which of two sets in the right-hand side of

$$\bar{C}_{\tilde{B}}(\tilde{\sigma} G_{\tilde{B}}) = \bar{C}_{\tilde{B}_R}\bar{C}_{\tilde{B}_L}(\tilde{\sigma} H) \cup \bar{C}_{\tilde{B}_R}\bar{C}_{\tilde{B}_L}(\tilde{\sigma}\tilde{\tau} H)$$

is nonempty. That information is now available from the computation of $\bar{C}_{B_L}$ with $\sigma[B_L]$. Thus we assume that the earlier computation recorded one bit at each node, $\tilde{B}$, directing the expanding process to $\tilde{\sigma}$ or $\tilde{\sigma}\tilde{\tau}$, and eliminating any need for the other computation. Let $p(\tilde{B})$ denote the number of nonfacile nodes in the pruned tree rooted at $\tilde{B}$. The expansion of $\alpha[B_L]$ to B requires at most one B_R-product at each of $p(B_L)$ nodes; this is done within $O(p(B_L)q(B_R))$ steps.

A second, similar problem arises in Cases 2 and 3 when the output from the call to $\bar{C}_{B_R}$ must be expanded, finally, to B. As above, this is done within $O(p(B_R)q(B_L))$ steps. The output from the call, in Case 1.5, requires no expanding since $\sigma(B \backslash \Delta(B)) \cap Q = \varnothing$.

Algorithm 3 is a modification of New Algorithm 1, assuming the blocks, groups, τ's, are given. It solves the following "localized" version of Problem V.

Problem VI. Fix a 2-group $G \leq \mathrm{Sym}(A)$, where A is a colored set. Given a G-stable subset, B, of A, Tree(B, G) together with entry groups at each node, and a local representative $\sigma[B]$ for some (otherwise unspecified) bijection σ: $A \to A'$, where A' is a second colored set. Compute

$$F(B, \sigma[B], G) = \begin{cases} \text{"}\varnothing\text{,"} & \text{if } \bar{C}_B(\sigma G) = \varnothing, \\ \text{a local representative } \bar{\sigma}[B] \text{ of some } \bar{\sigma} \in \bar{C}_B(\sigma G), & \text{otherwise.} \end{cases}$$

In the recursions below, recall that Tree(B, G) induces a suitable Tree($\tilde{B}$, $G_{\tilde{B}}$) at each node $\tilde{B}$ in B, and the entry groups are inherited by the subtree.

Algorithm 3 (*for Problem VI*).

Case 1a. ($\exists i$: $|B \cap Q_i| \neq |\sigma(B) \cap Q_i|$): $F(B, \sigma[B], G)$:= "$\varnothing$"; **return**

Case 1b. ($\exists i$: $B \cup \sigma(B) \subseteq Q_i$): $F(B, \sigma[B], G) := \sigma[B]$; **return**

Case 1.5. (B is facile):
$F(B, \sigma[B], G) := F(\Delta(B), \sigma[\Delta(B)], G_{\Delta(B)})$
(* in this situation $\sigma[\Delta(B)] = \sigma[B]$ *);
return

Case 2. (B is intransitive):
recursively compute $\Lambda = F(B_L, \sigma[B_L], G)$;
if Λ = "$\varnothing$" **then** ($F(B, \sigma[B], G)$:= "$\varnothing$"; **return**);
$\alpha[B]$:= the expansion of Λ to B;
recursively compute $\Lambda 1 = F(B_R, \alpha[B_R], G_{B_R})$;
if $\Lambda 1$ = "$\varnothing$" **then** ($F(B, \sigma[B], G)$:= "$\varnothing$"; **return**);
$F(B, \sigma[B], G)$:= the expansion of $\Lambda 1$ to B;
return

Case 3. (B is transitive):
recursively compute $\Lambda = F(B_L, \sigma[B_L], G_{B_L})$;
if $\Lambda \neq$ "$\varnothing$" **then**
begin
$\alpha[B]$:= the expansion of Λ to B;
recursively compute $\Lambda 1 = F(B_R, \alpha[B_R], G_{B_R})$;
if $\Lambda 1 \neq$ "$\varnothing$" **then**
($F(B, \sigma[B], G)$:= the expansion of $\Lambda 1$ to B; **return**)
end;
retrieve $\tau \in G_B \backslash G_{B_L}$;
recursively compute $\Lambda 2 = F(B_L, \sigma\tau[B_L], G_{B_L})$;
if $\Lambda 2 \neq$ "$\varnothing$" **then**
begin
$\beta[B]$:= the expansion of $\Lambda 2$ to B;
recursively compute $\Lambda 3 = F(B_R, \beta[B_R], G_{B_R})$;
if $\Lambda 3 \neq$ "$\varnothing$" **then**
($F(B, \sigma[B], G)$:= the expansion of $\Lambda 3$ to B; **return**)
end;
$F(B, \sigma[B], G)$:= "$\varnothing$";
return

We employ Algorithm 3, at stage r in the computation of $\mathrm{Iso}_{e_1,e_2}(X^1, X^2)$, with $B = B_r$, $G = \mathrm{Aut}_{e_1}(X_r^1)$, and $\sigma = \sigma_r \in \mathrm{Iso}_{e_1,e_2}(X_r^1, X_r^2)$. The output is either "$\varnothing$" or $\alpha[B_r]$ for some $\alpha \in \bar{C}_{B_r}(\sigma_r \mathrm{Aut}_{e_1}(X_r^1))$. If the former, there is no isomorphism from X^1 to X^2 sending e_1 to e_2. If the latter, it remains to determine $\sigma_{r+1} \in \mathrm{Iso}_{e_1,e_2}(X_{r+1}^1, X_{r+1}^2)$. For that, a final pass over the pruned tree $\mathrm{Tree}_p(B_r, G_r)$, as in the expansion process, extends $\alpha[B_r]$ to liftable $\sigma_{r+1} \in \mathrm{Iso}_{e_1,e_2}(X_r^1, X_r^2)$. Since this requires at most one product of permutations on an $O(n)$ element set at each of $p(B_r)$ nodes, it is done within $O(np(B_r))$ steps.

We analyze the isomorphism test in the next section.

10. *The Time Bound for the Isomorphism Test*

We test isomorphism through the reduction of Section 8 and the discussion of Section 9. The preliminary gathering of $\mathrm{Aut}_{e_1}(X^1)$ information is done in $O(n^3)$ steps. We express the time requirement of Algorithm 3 as a function of the number of nonfacile nodes and the number of colored leaves in Tree(B, G). That is, let $t(p, q)$ be the worst case running time for any Tree(B, G) with $p(B) \le p$, $q(B) \le q$.

LEMMA 14. *With t as above,* $t(p, q) = O(pq \log pq)$.

PROOF. One may assume that $t(1, q) = t(p, 0) = 1$ (note that $p = 1$ happens only at a leaf). Suppose that $p(B) > 1$, $q(B) > 0$. Set $p_1 = p(B_L)$, $p_2 = p(B_R)$, $q_1 = q(B_L)$, $q_2 = q(B_R)$, so that $q(B) = q_1 + q_2$ and, if B is nonfacile, then $p(B) = p_1 + p_2 + 1$. It suffices to prove the bound for nonfacile B for then, in the facile case, the cost is $O(q(B))$ + the cost of the tree rooted at $\Delta(B)$. Thus we redefine $t(p, q)$ as the worst case time over all nonfacile B with $p(B) \le p$, $q(B) \le q$. Suppose that B, G is a "worst case" among these, that is, the time demanded by Tree(B, G) is $t(p(B), q(B))$. Taking all constants to be 1, we have the alternatives

$$t(p(B), q(B)) \le \begin{cases} 2 \sum_{i=1}^{2} t\left(p_i, \dfrac{q(B)}{2}\right) + (p_1 + p_2)\dfrac{q(B)}{2} + q(B) \\ \\ \sum_{i=1}^{2} t(p_i, q_i) + p_1 q_2 + p_2 q_1 + q(B). \end{cases}$$

Here, we have associated the cost at B to its first nonfacile descendents on the left and right. The first inequality holds when B is transitive and $q_1 = q_2 = q(B)/2$ and the second otherwise. The final $q(B)$ accounts for any B-product. It also recognizes the time for tests in Cases 1a and 1b, including those at facile sons, as well as those at the sons if B is transitive and the colors do not split evenly between B_L and B_R (which is why that transitive subcase can be covered by the second inequality). The terms $(p_1 + p_2)(q(B)/2)$ and $p_1 q_2 + p_2 q_1$ account for computation of expansions (though there may be *two* expansions that contribute to the $p_1(q(B)/2)$ term recall that we have taken all constants to be 1). The lemma follows by induction. □

This is the essence of

LEMMA 15. *Assuming that all the blocks and groups from the computation (as in Sections 5–7) of* $\mathrm{Aut}_{e_1}(X^1)$ *are given, the set*

$$\mathrm{Iso}_{e_1 e_2}(X^1, X^2) = \sigma \mathrm{Aut}_{e_1}(X^1)$$

of isomorphisms from X^1 *to* X^2 *that map* e_1 *to* e_2, *is computable in time* $O(n^2 \log n)$.

PROOF. The initial formation of $\sigma[B_r]$ from σ_r is done in $O(|B_r|) = O(m_r^2)$ steps, for a total, over all stages, of only $O(n^2)$. As indicated, the computation of $\sigma_{r+1}[B_r]$ in stage r takes $t(p(B_r), q(B_r)) = O(p(B_r)q(B_r)\log p(B_r)q(B_r))$ steps. The lifting to $\sigma_{r+1} \in \mathrm{Iso}_{e_1, e_2}(X^1_{r+1}, X^2_{r+1})$ is done in $O(np(B_r))q(B_r))$ steps. The result follows by Lemmas 10 and 11. □

Hence,

THEOREM 3. *Isomorphism of n-vertex trivalent graphs can be tested in time* $O(n^3 \log n)$. *The algorithm determines the set of all isomorphisms.*