A SUBEXPONENTIAL ALGORITHM

FOR

TRIVALENT GRAPH ISOMORPHISM

Merrick Furst[1]

John Hopcroft[2]

Eugene Luks[3]

TR 80-426

[1]Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pa.

[2]Department of Computer Science, Cornell University, Ithaca, N.Y.

[3]Department of Mathematics, Bucknell University, Lewisburg, Pa.

This research was supported in part by ONR grant N00014-76-C-0018 and an NSF Small College Faculty Allowance.

1. Introduction

The best of the known algorithms for testing isomorphism of general undirected graphs have running times exponential in n, the number of vertices. To increase the efficiency of testing isomorphism, heuristics are often used. Typically, these heuristics partition the vertices into classes with given properties and thereby reduce the number of bijections that must be checked as possible isomorphisms [3, 11]. Even successful classification of the vertices into disjoint sets of size ≤ 3 leaves an exponential number of maps to check.

L. Babai [3] explored algorithms for testing isomorphism of graphs whose vertices have been partitioned into classes of size k or less, for some fixed k. He was able to exhibit a polynomial-time Las Vegas algorithm, i.e. an algorithm in $R \cap coR$, to solve this problem. He also observed that no subexponential deterministic algorithm was known. Our first result provides a polynomial-time algorithm for this problem. Specifically, we exhibit an algorithm to test color preserving isomorphism of two vertex-colored graphs in which each color class is of size k or less. In addition, we obtain generators for the automorphism group of such a graph in polynomial time.

A group G acting on $\{1,\ldots,n\}$ is said to be <u>recognizable in time T(n)</u> if the question "is x in G?" is decidable in T(n) time. A tower of groups $G = G_0 \supset G_1 \cdots \supset G_r = I$ is called <u>polynomially accessible</u> if

(i) generators are known for G,

(ii) each G_i is recognizable in polynomial time,

and (iii) $|G_i/G_{i+1}| \leq p(n)$, for some polynomial p.

Our second result is that the size of, and generators for, any group which is polynomially accessible via such a tower can be determined in polynomial time. This gives a polynomial-time algorithm for computing the automorphism group of a vertex-colored graph with bounded color multiplicity. It also solves, in polynomial time, the isomorphism testing problem for such graphs [3].

C. Hoffmann [9] exhibited a Las Vegas algorithm for determining the automorphism group of a trivalent cone graph in time $O(n^{c\log n})$ using a recursive application of Babai's algorithm. We improve this result and give a deterministic $O(n^{c\log n})$ algorithm for computing the automorphism group of a slightly more general structure.

Finally, we address the problem of testing isomorphism of trivalent graphs. The best previously known algorithm, attributed to G. Miller, runs in time $O(c^n)$ for some constant c. Testing trivalent graph isomorphism can be polynomial-time reduced to computing the automorphism group of a trivalent graph in which one edge is fixed. It is a well known, and easy to prove, theorem of Tutte's that such an automorphism group has order 2^k for some integer k. Exploiting the unique properties of groups with order a power of a prime, and using the improvements to Babai's and Hoffmann's algorithms, we derive a subexponential ($O(n^{c\log n})$) algorithm for trivalent graph isomorphism.

2. Polynomially Accessible Towers of Groups

A *permutation group* on $\{1,\ldots,n\}$ is a collection of 1-1 maps from $\{1,\ldots,n\}$ onto itself that forms a group. Let G be a group, $|G|$ stands for the order of G.

Let H be a subgroup of G. The symbol G/H stands for the collection of cosets of H in G, i.e., the collection of equivalence classes in which $x \equiv y$ if and only if $x^{-1}y$ is in H. From Lagrange's theorem we know that every coset of G/H has the same size and that
$|G| = |G/H| \times |H|$.

If $g_1,\ldots,g_k$ are permutations then $\langle g_1,\ldots,g_k\rangle$ stands for the group of all permutations formed by products of the g_i.

2.1 Towers of Groups

L. Babai [3] seems to have been the first researcher to suggest that computing the size of an unknown group might be facilitated by first trapping that group in a tower. He used this technique on the automorphism groups of vertex-colored graphs of bounded color multiplicity and the automorphism groups of graphs with bounded eigenvalue multipicity. By trapping such groups in polynomially accessible towers he was able to calculate their size in polynomial time using coin tossing algorithms. Our first result is a polynomial time deterministic algorithm for this problem.

Theorem 1.1: Let $G = G_0 \supset \cdots \supset G_r = I$ be a polynomially accessible tower of groups. Coset representatives for G_i/G_{i+1}, $i=0,\ldots,r-1$, can be determined in polynomial time.

__Proof__ : Let $\max_i |G_i/G_{i+1}| = k$. We will construct a table with r rows, numbered 0 to r-1, and k columns whose ith row is a set of right coset representatives for G_i/G_{i+1}. The group G can be expressed as

$$G = G_0 = (G_0/G_1)G_1,$$

or, $G = (G_0/G_1)(G_1/G_2)\cdots(G_{r-1}/G_r).$

Therefore, any element g in G can be written in the form $g = a_0a_1\cdots a_{r-1}$, where a_i is an element of G_i/G_{i+1}.

The table should have the following property: g is in G if and only if g can be expressed as $a_0a_1\cdots a_{r-1}$ where a_i is a member of the ith row. A permutation g is in __canonical form__ when it is written this way.

We start with a table whose rows contain only the identity element. The procedure sift(x), defined below modifies the table so that the element x from G can be written in canonical form.

```
sift(x):
        i ← 0
        while (i ≠ r-1 and there is a y in row i such that y^-1 x is in G_{i+1})
            do
                i ← i + 1
                x ← y^-1 x
            od
        if x is not the identity then insert x in row i
```

As an example of how sift works, suppose the table for the tower $G_0 \supset G_1 \supset G_2 \supset G_3 = I$ at some point looks like Figure 2.1.

$$(G_0=)\quad |\underline{\quad 1 \quad}|\, G_1 \;+\; |\underline{\quad a \quad}|\, G_1 \;+\; |\underline{\qquad\quad}|\, G_1$$
$$(G_1=)\quad |\underline{\quad 1 \quad}|\, G_2 \;+\; |\underline{\qquad\quad}|\, G_2 \;+\; |\underline{\qquad\quad}|\, G_2$$
$$(G_2=)\quad |\underline{\quad 1 \quad}|\, G_3 \;+\; |\underline{\qquad\quad}|\, G_3 \;+\; |\underline{\qquad\quad}|\, G_3.$$

Figure 2.1 State of table after sift(a).

Consider sift(b) for some b in G_0 but not in G_1. If $a^{-1}b$ is in G_1, and $a^{-1}b$ is not in G_2, then after the call sift(b) the table would look like Figure 2.2, at which point $b = a(a^{-1}b)$ is expressible in canonical form.

1	a	
1	$a^{-1}b$	
1		

Figure 2.2 State of table after sift(b).

If we had time to sift every element of G, then we would be sure that every element of G could be expressed in canonical form.

At this point we make a key observation: all of the coset representatives have been found if and only if

(1) Each generator can be written in canonical form,

and (2) Each product of a pair of representatives in the table can be written in canonical form.

Since the only elements ever sifted into the table are from G we need only verify that when (1) and (2) are satisfied any g in G can be written in the canonical form. Let g be an element of G. Write g as a product of generators and write each generator in canonical form. If this product is not in canonical form, we use (2) to rewrite it.

By using (2) we can take an adjacent pair of representatives x,y in the string representing g and, if x comes from a higher numbered row than y, rewrite xy in canonical form. This has the effect of moving an element from a lower numbered row past an element of a higher numbered row to the left in the string. Moving all the row 0 representatives to the left, then all the row 1 representatives, and so on, we can put the string in canonical form. It is important to note here that writing xy in canonical form does not require the introduction of any elements from rows numbered less than the one y comes from.

The whole algorithm can be described as

Step 1. Sift all the generators.

Step 2. Sift the product xy for every pair x,y in the table.

The timing analysis is straightforward. The number of coset representatives in the table is at most m, some polynomial in n. The number of calls to sift is at most m^2 and each call to sift takes roughly $k(r-1)p(n)$ time, where $p(n)$ is a polynomial that bounds the amount of time to test membership in any G_i. Therefore, the running time is polynomial in n.

□

An immediate corollary is an algorithm much like the one proposed by C. Sims [12].

<u>Corollary</u> 1.2: Let G be a group of permutations generated by $g_1,\dots,g_k$. Let G_i be the subgroup of G that fixes $1,\dots,i$. Then coset representatives for G_i/G_{i+1} can be determined in polynomial time. Furthermore, this gives a polynomial-time decision procedure for testing membership in G and for determining the order of G.

<u>Proof</u> : Use the algorithm described in the previous theorem to compute the coset representatives. The only modification is that to test whether x and y are inequivalent members of G_i/G_{i+1} just check that they map i+1 to different places.

Once the coset representatives have all been found, testing membership is not hard. To determine if x is an element of G, run the procedure sift with argument x. If x can be written in canonical form, then x has been written as a product of generators. If x cannot be written in cannonical form, then x is not in G.

The order of G is the product of the sizes of G_i/G_{i+1}.

□

3. <u>Computing the Intersection of Two Groups</u>

Consider the problem of computing the automorphism group of a graph G. Suppose it were possible to partition the edges of G into two classes, A and B, in such a way that every automorphism of G fixed, setwise, these classes. Let X_A and X_B be the automorphism groups that preserve the A edges and the B edges of G respectively. The permutations in the intersection of X_A and X_B are exactly the automorphisms of G.

If we had a fast algorithm for computing the intersection of two groups, then we could compute the automorphism group of G inductively. At present we do not know how to take the intersection of an arbitrary pair of groups in polynomial time. Using the following theorem, however, we can get the intersection of two groups under special circumstances.

Theorem 3.1: Let G and H be any two polynomial-time recognizable groups. Let S be a group for which generators are known. If S contains both G and H, and there are two polynomially accessible towers, one from S through G to I, and the other from S through H to I, then generators for $G \cap H$ can be found in polynomial time.

Proof : Let $S = H_0 \supset \cdots \supset H_r = H \supset H_{r+1} \supset \cdots \supset H_{s-1} = I$, and $S = G_0 \supset \cdots \supset G_p = G \supset G_{p+1} \supset \cdots \supset G_{q-1} = I$ be the two towers. Construct an $s \times q$ table whose i,jth entry is the group $G_i \cap H_j$. Each entry is a recognizable group since it is the intersection of recognizable groups.

Both $|H_j/H_{j+1}|$ and $|G_i/G_{i+1}|$ are bounded by some polynomial p(n). Consider the two groups $G_i \cap H_j$ and $G_i \cap H_{j+1}$. Let a and b be distinct coset representatives of $X = (G_i \cap H_j)/(G_i \cap H_{j+1})$. The elements a and b are both from the group H_j. Furthermore, if $a^{-1}b$ were an element of H_{j+1}, it would also be an element of $G_i \cap H_{j+1}$. Since a and b are from different cosets of X it follows that $a^{-1}b$ is not in H_{j+1}. Therefore, a and b are distinct coset representatives of H_j/H_{j+1}. Hence

$|(G_i \cap H_j)/(G_i \cap H_{j+1})|$ is less than or equal to $|H_j/H_{j+1}| \le p(n)$. Similarly, $|(G_i \cap H_j)/(G_{i+1} \cap H_j)| \le p(n)$.

Let P be any path in the table beginning at S, moving only one row down or one column across at a time, passing through $G \cap H$, and ending at I. This path P describes a polynomially accessible tower of groups from S through $G \cap H$ to I.

For example, the tower

$S \supset H_1 \supset H_2 \supset \cdots \supset H \supset G_1 \cap H \supset \cdots \supset G_p \cap H \supset G \cap H \supset G_{p+1} \cap H \supset \cdots \supset I$ can be used to get generators for $G \cap H$ in polynomial time.

$\square$

4. The Sylow Theorems, 2-Groups, and Imprimitivity

The automorphism group of a trivalent graph with one edge fixed is a 2-group, i.e. it has order 2^k for some k. Groups of order p^k, where p is a prime, have special structure which we will exploit later. In this section we state some of the standard theorems. For more detail we refer the reader to [8].

Theorem (Sylow): If G is a group and 2^k is the largest power of two dividing $|G|$ then G contains a subgroup of order 2^k. Such a subgroup is called a 2 Sylow subgroup (2SSG).

Theorem (Sylow): All 2SSG's of a group G are isomorphic.

<u>Theorem</u> (Sylow): Let S be a subgroup of G. If S is a 2-group then S is contained in a 2SSG of G.

The 2SSG's of S_n have the following characterization in terms of the automorphism group of binary trees.

<u>Theorem</u> 4.1: Let S be any 2 Sylow subgroup of S_n, where n is a power of 2. The group S can be viewed as the action, on the leaves, of the automorphism group of a complete binary tree whose leaves are the vertices 1,...,n in some order.

<u>Proof</u> : Draw the graph of a complete binary tree whose leaves are the vertices 1,...,n in some order. Label the internal nodes of the tree by the permutation of the leaves obtained when the node's children are interchanged. Let the group of the tree be the group generated by these permutations. Since all 2SSG's of S_n are isomorphic, and the group of the tree is a subgroup of S_n, all we need show is that the order of this group is the largest power of two dividing (n!).

The group of the tree has order 2 to the power of the number of internal nodes, i.e. 2^{n-1}. The factors of n! divisible by 2 are 1×2, 2×2,...,$(n/2) \times 2$. Thus the highest power of two dividing n! is (n/2) + r where r is the largest power of two dividing (n/2)!. By induction, k = (n-1) is the largest number such that 2^k divides (n!) for n a power of 2. From this we conclude that every 2SSG of S_n can be viewed as the automorphism group of a binary tree whose leaves are {1,...,n} in some order.

□

If n is not a power of two, a 2SSG of S_n can be shown to be the group induced by automorphisms of a forest of full binary trees.

Example : Any 2SSG of S_6 is isomorphic to the group induced by the automorphisms of the trees in Figure 4.1.

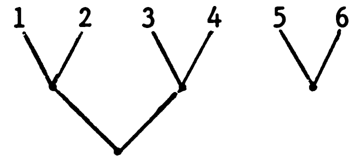

Figure 4.1 A 2SSG of S_6.

4.1 Imprimitive Groups

Let G be a permutation group on $\{1,\ldots,n\}$. Suppose $\{1,\ldots,n\}$ can be partitioned into disjoint sets $S_1,\ldots,S_m$ such that every permutation in G either maps all the elements of S_i onto S_i or onto the elements of another S_j. If $S_1,\ldots,S_m$ can be chosen in a non-trivial way, i.e. not just one universal set or n singleton sets, then G is called _imprimitive_, and the sets S_i are called _domains_ or _sets of imprimitivity_.

Every 2SSG of S_n has n/2 sets of imprimitivity of size 2, n/4 of size 4, etc. These correspond to leaves of subtrees of height 1,2,..., respectively. Since every 2-group of permutations is contained in a 2SSG of S_n, each is imprimitive.

4.2 Computing with 2-groups

Let G be some 2-group. In this section we show how to find Syl(G), a 2SSG of S_n containing G, in polynomial time. We then show how to determine a polynomially accessible tower from Syl(G) through G to I. Using this result and the intersection theorem, it will be possible to calculate the intersection of any two 2-groups that are subgroups of a single 2SSG.

Theorem 4.2: Let $G = \langle g_1, \ldots, g_k \rangle$ be a 2-group acting on $\{1, \ldots, n\}$. Syl(G), a 2 Sylow subgroup of S_n containing G, can be found in polynomial time.

Proof : Since G is a 2-group and is therefore contained in some 2SSG of S_n, G must be imprimitive and have domains of imprimitivity contained in those of the 2SSG. In particular, G must have n/2 domains of imprimitivity of size 2, corresponding to the leaves of subtrees of height 1 in the tree representation of the 2SSG.

Such a collection of sets can be located quickly. First find a pair (i_0, j_0) whose orbit under G contains only disjoint pairs. To find the orbit of a pair, use a transitive closure algorithm on the action of the generators. The collection of pairs in the orbit of (i_0, j_0) form domains of imprimitivity of size 2 for G. Since the group G acts as a 2-group on the vertices not in this orbit, this algorithm can be applied repeatedly until all the vertices are paired up into domains of imprimitivity of size 2.

Since G is a 2-group, the action of G on this collection of pairs must also be a 2 group, say G_1. A cycle x of non-power-of-two length in G_1 would imply the existence of a non-power-of-two length cycle in G.

By determining a collection of sets of imprimitivity of size two in G_1 we also determine domains of imprimitivity of size 4 in G. Continuing recursively, we can obtain partitions of $\{1,\ldots,n\}$ into sets of imprimitivity of sizes $2,4,8,\ldots,2^{n-1}$ with respect to G. In turn, these domains of imprimitivity specify a unique 2SSG with the same domains. This 2SSG, call it Syl(G), contains all permutations on $\{1,\ldots,n\}$ that respect its domains of imprimitivity. Since every permutation of G respects its domains of imprimitivity, G is contained in Syl(G).

□

Having obtained Syl(G), a 2SSG of S_n containing G, we now want to determine a polynomially accessible tower from Syl(G) through G to I. The top portion of the tower, $Syl(G) = H_r \supset H_{r-1} \supset \cdots \supset H_1 \supset G$, will be defined by letting $H_i = \langle G, a_1, \ldots, a_i \rangle$, where the a_i are elements of Syl(G). These a_i must be chosen with some care so that $|H_{i+1}/H_i|$ is bounded by a polynomial, and $H_r = Syl(G)$.

Let S be any 2-group. It can be shown [8] that there is a tower of groups $S = S_r \supset S_{r-1} \supset \cdots \supset S_0 = I$ in which $|S_i/S_{i-1}| = 2$. For the special case in which $S = Syl(G)$ is a 2SSG of S_n, it is possible to find a sequence $a_1,\ldots,a_r$ such that S_i can be taken to be $\langle a_1,\ldots,a_i \rangle$.

<u>Lemma</u> 4.3: Let S be a 2 Sylow subgroup of S_n, $n = 2^k$. In polynomial time it is possible to determine a sequence of elements $a_1,\ldots,a_r$ of S such that

(I) $(a_i)^2 = 1$,

(II) a_i commutes with every element of $S/\langle a_1,\ldots,a_{i-1}\rangle$,

and (III) $S = \langle a_1,\ldots,a_r\rangle$.

<u>Proof</u> : The proof is by induction on the height of the binary tree representing S. When n is 2, the height of S is 1 and S is the tree in Figure 4.2, where a is the permutation (1 2). For this tree the sequence would be $a_1 = a$.

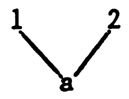

Figure 4.2 The 2SSG of S_2.

Suppose S is of height k and it is known how to determine such a sequence for any tree of height k-1. The tree representing S can be viewed as having a root with two subtrees of height k-1 attached, Figure 4.3.

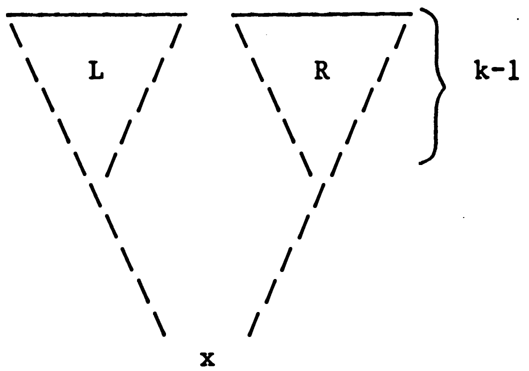

Figure 4.3 The 2SSG S.

By the induction hypothesis, we find a sequence $a_1,\ldots,a_r$ for the left subtree, L. The right subtree, R, is isomorphic to L using the map $m:L \to R$ in which $m(y) = xyx$. Therefore, we can take the sequence $b_1,\ldots,b_r$, with $b_i = xa_ix$, to be a sequence for R.

The claim is that the sequence $a_1b_1,\ a_1,\ a_2b_2,\ a_2,\ldots,a_rb_r,\ a_r,\ x$ satisfies (I), (II), and (III) for S.

Since a_i and b_i operate on disjoint sets of vertices, $a_ib_j = b_ja_i$, for any i and j. It follows that each member of this sequence satisfies condition (I). We must demonstrate two things to prove that condition (II) is satisfied.

(1) The element a_ib_i commutes with each of the generators $a_i, a_{i+1}b_{i+1},\ldots,x$ of $S/\langle a_1b_1,\ldots,a_{i-1}\rangle$:

Since a_i and b_i commute, and $a_1 = a_2 = \cdots = a_{i-1} = b_1 = \cdots = b_{i-1} = 1$ in $S/\langle a_1b_1,\ldots,a_{i-1}\rangle$, it follows that a_ib_i commutes with each generator, except perhaps x. By algebra,

$$\begin{aligned} xa_ib_i &= xa_i(xa_ix) \\ &= (xa_ix)a_ix \\ &= b_ia_ix \\ &= a_ib_ix. \end{aligned}$$

Therefore, a_ib_i commutes with every elememt of $S/\langle a_1b_1,\ldots,a_{i-1}\rangle$.

(2) The element a_i commutes with each of the generators $a_{i+1}b_{i+1},\ldots,a_rb_r,x$ of $S/\langle a_1b_1,\ldots,a_ib_i\rangle$:

It is certainly the case that a_i commutes with all the generators with the possible exception of x. Thus all we need to show is that $a_ix = xa_i$. By algebra,

$$\begin{aligned} a_ix &= a_i(a_ib_i)x, \qquad \text{since } a_ib_i \equiv 1 \\ &= b_ix \\ &= xa_i. \end{aligned}$$

This shows that a_i commutes with every element of $S/\langle a_1b_1,\ldots,a_ib_i\rangle$.

Clearly condition (III) is satisfied. Therefore, the sequence a_1b_1, $a_1,\ldots,a_rb_r$, x satisfies (I),(II), and (III).

□

In the event that n is not a power of two, Syl(G) is represented by a forest of full binary trees. The direct product of the automorphism groups of these trees is Syl(G). Therefore, a sequence for Syl(G) is the concatenation of sequences, one for each of the trees.

<u>Theorem</u> 4.4: Let $G = \langle g_1,\ldots,g_k \rangle$ be a subgroup of a 2SSG S of S_n. Let $a_1,a_2,\ldots,a_r$ be a sequence satisfying conditions (I), (II), and (III) of Lemma 4.3. The groups $G \subset H_1 \subset \cdots \subset H_r = S$, where $H_i = \langle G, a_1, a_2,\ldots, a_i \rangle$, form a recognizable tower. Furthermore, the index of H_i in H_{i+1} is no larger than 2.

<u>Proof</u> : Each of the groups is recognizable since generators for it are available. If we show that a_{i+1} normalizes H_i we will be done since this would imply that H_{i+1} equals either H_i, or $(H_i + a_{i+1}H_i)$.

Consider the homomorphism f_i from S into $\overline{S} = S/\langle a_1,\ldots,a_i \rangle$. Let $\overline{H}_i$ be the image of H_i in $\overline{S}$ under f_i. The element a_{i+1} commutes with every element of $\overline{S}$ so a_{i+1} normalizes $\overline{H}_i$ in $\overline{S}$. Since H_i contains the kernel, $\langle a_1,\ldots,a_i \rangle$, of f_i, the normalizer of H_i in S contains the preimage, under f_i, of the normalizer of $\overline{H}_i$ in $\overline{S}$. Therefore, a_{i+1} is in the normalizer of H_i.

□

Since G is a group of permutations, there is a tower of recognizable groups $G \supset G_1 \supset \cdots \supset I$ where G_i is the subgroup of G that fixes vertices $1,\ldots,i$. From this we get the following important corollary.

<u>Corollary</u> 4.5: If G is a 2-group with generators $g_1,\ldots,g_k$, a polynomially accessible tower from Syl(G) through G to I can be located in polynomial time.

The major theorem of this section says that given two 2-groups which are subgroups of a common 2SSG, generators for their intersection can be found in polynomial time.

Theorem 4.6: Let $G = \langle g_1,\ldots,g_k \rangle$, $H = \langle h_1,\ldots,h_r \rangle$ be two 2 groups, and let S, a 2SSG containing them, be given. Generators for $G \cap H$ can be determined in polynomial time.

Proof : Use Corollary 4.5 to construct two polynomially accessible towers, one from S through G to I, the other from S through H to I. Use Theorem 3.1 to calculate the generators for $G \cap H$.

□

5. Cone Graphs and Hoffmann's Algorithm

Definition: A cone graph on the vertices $1,\ldots,n$ is the graph of a complete binary tree with leaves $1,\ldots,n$, together with a hypergraph on $1,\ldots n$. If G and H are cone graphs of the same size, their union G + H is called a double cone graph. The vertices of cone graphs and double cone graphs can be partitioned into classes called levels corresponding to distance from the leaves. The leaves are at level 0 and the roots are at level logn. Figure 5.1 is an example of a cone graph.

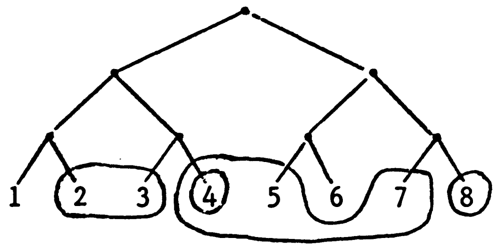

Figure 5.1 A cone graph with 4 hyperedges

We prove the following lemma in a fashion similar to the one used by C. Hoffmann in [9]. Using this lemma we will be able to compute the automorphism group of a cone graph in deterministic time $O(n^{c\log n})$, an improvement over Hoffmann's probabilistic result.

Lemma 5.1: Let G + H be a double cone graph whose hyperedges are no more than k-ary for some fixed constant k. The automorphism group of G + H mapping roots onto roots can be determined in time $O(n^{c\log n})$.

Proof : Let z be any 1-1 map from the vertices of G to the vertices of H that preserves tree edges. For each i, define the group $U_i(G)$ ($U_i(H)$) to be the automorphism group of the cone graph G (H) that preserves tree edges and fixes vertices at levels i and above. Let $U_i(G,H,z)$ be the group $U_i(G) \times U_i(H) \times \langle z\rangle$. An element (g,h,s) in $U_i(G,H,z)$ is the permutation ghs on G + H. We define $A_i(G,H,z)$ to be the subgroup of $U_i(G,H,z)$ that preserves the hyperedges of G + H.

The automorphism group of G + H is thus $A_{\log n}(G,H,z)$. In order to get generators for this group we obtain generators for the groups $A_i(G,H,z)/A_{i-1}(G,H,z)$. Since $A_i(G,H,z) \supset A_{i-1}(G,H,z)$ the union of the generators for the quotient groups generate $A_{\log n}(G,H,z)$.

By a classical isomorphism theorem [6], $A_i(G,H,z)/A_{i-1}(G,H,z) \cong A_i(G,H,z)U_{i-1}(G,H,z)/U_{i-1}(G,H,z)$. Each of these groups describes the action of A_i on the level i-1 vertices. It is the latter group that we trap in the tower $U_i/U_{i-1} \supset W_1 \supset \cdots \supset W_t = A_i(G,H,z)U_{i-1}/U_{i-1} \supset V_1 \supset \cdots \supset V_r = I$, where U_i stands for $U_i(G,H,z)$. The groups V_j are just subgroups of W_t fixing successively more vertices. The W_j are a little harder to describe. To define them we have to construct a sequence of graphs.

Let X_0 be the binary trees of G + H without any of the hyperedges. The first group in the tower, $U_i(G,H,z)/U_{i-1}(G,H,z)$, describes the action of the automorphism group of X_0 on level i-1 vertices. Each element of this group can be represented by a 0-1 assignment to level i vertices together with a 0 or a 1 indicating whether the trees G and H should be interchanged according to z. A 1 assigned to a level i vertex means interchange his sons, a 0 means leave them alone.

Pick a vertex v from among the level i vertices in G. Let X_1 be X_0 plus all of the unary hyperedges on leaves of the subtrees rooted at v in G and z(v) in H. Let X_2 be X_1 plus the unary edges from the subtrees rooted at w and z(w) for some other level i vertex w in G. Continue in this fashion until the graphs $X_0 \subset \cdots \subset X_{m_i}$, where m_i is the number of level i vertices in G, have been defined.

After processing the unary hyperedges, process the binary edges, then the ternary edges, etc. until all of the hyperedges of G + H have been incorporated into a graph, X_t. This yields a sequence of graphs $X_0 \subset X_1 \subset \cdots \subset X_t = G + H$. Each X_j is the union of two cone graphs we can call G_j and H_j. The groups $W_1 \supset \cdots \supset W_t$ can now be described as $W_j = A_i(G_j,H_j,z)U_{i-1}(G,H,z)/U_{i-1}(G,H,z)$.

We do not know whether the tower $U_{i+1}/U_i \supset W_1 \supset \cdots \supset W_{t-1} \supset A_{i+1}U_i/U_i \supset V_1 \supset \cdots \supset V_r = I$ is polynomially accessible. We can show that the index of each group in the next is bounded by a polynomial in n, but we don't know whether the W_j are recognizable in polynomial time.

Hoffmann used Babai's coin tossing algorithm to recognize W_j in time $O(n^{ci})$. Using our improvement to Babai's algorithm we can recognize elements of W_j in deterministic time $O(n^{ci})$. It is because of this step that the algorithm computing $A_{\log n}$ runs in time $O(n^{c\log n})$.

The graph X_{j+1} is X_j together with d-ary hyperedges between the leaves of subtrees rooted at some vertices $a_1,\ldots,a_d$ and $z(a_1),\ldots,z(a_d)$ on level i. Adding these constraints to the automorphism group of X_j can at most restrict the allowable 0-1 assignments to 2d level i+1 vertices. Therefore, the size of W_j/W_{j+1} is at most 2×2^{2d}, which is less than the constant 2^{2k+1}.

Now we turn to the problem of determining whether two elements x and y of W_{j-1} are in the same coset of W_{j-1}/W_j. This is the case if and only if $f = x^{-1}y$ is in $A_i(G_j,H_j,z)U_{i-1}(G,H,z)$. The permutation f is in the product of these groups if and only if there is a u in $U_{i-1}(G,H,z)$ such that uf is an automorphism in $A_i(G_j,H_j,z)$.

The graph $f(G_j + H_j)$ consists of two cones, $f(G_j)$ and $f(H_j)$. For u in $U_{i-1}(G,H,z)$, uf is in $A_i(G_j,H_j,z)$ if and only if the graph

$uf(G_j) + uf(H_j)$ is isomorphic to $G_j + H_j$. There are only two ways this isomorphism can work and they correspond to the following two cases.

Case 1: $uf(G_j)$ is isomorphic to G_j and $uf(H_j)$ is isomorphic to H_j.

Such a u exists if and only if there is a u_G in $A_{i-1}(G_j,f(G_j),f)$ that maps G_j onto $f(G_j)$ and a u_H in $A_{i-1}(H_j,f(H_j),f)$ that maps H_j onto $f(H_j)$.

Case 2: $uf(G_j)$ is isomorphic to H_j and $uf(H_j)$ is isomorphic to G_j.

Such a u exists if and only if there is a u_G in $A_{i-1}(G_j,f(H_j),fz)$ that maps G_j onto $f(H_j)$ and a u_H in $A_{i-1}(H_j,f(G_j),fz)$ that maps H_j onto $f(G_j)$.

In either case, testing for the existence of a u in $U_{i-1}(G,H,z)$ requires the construction of two A_{i-1}'s. To construct A_i, order n^d, for some constant d, tests of memebership in W_j must be carried out. Thus, if T(i) is the time to construct A_i, $T(i) \le 4n^d T(i-1) \le O(n^{ci})$.

Therefore, constructing $A_{\log n}(G,H,z)$ by this method takes time $O(n^{c\log n})$, for some constant c.

□

To compute generators for the automorphism group of a single cone graph G whose hyperdeges are at most k-ary, for some constant k, we use the following theorem.

<u>Theorem</u> 5.2: Let G be a cone graph whose hyperedges are at most k-ary. The automorphism group A_G of G mapping the root onto itself can be determined in time $O(n^{c\log n})$.

Proof : Find A, the automorphism group of G + G, in time $O(n^{c\log n})$. Construct the polynomially accessible tower $A \supset A_G \supset A_1 \supset \cdots \supset A_n = I$, in which the A_i fix successively more vertices. Using this tower obtain generators for A_G.

□

6. An $O(n^{c\log n})$ Trivalent Graph Isomorphism Algorithm.

Having established all the group theoretic tools and algorithms of the previous sections we can at last describe how to determine generators for the automorphism group of a trivalent graph G with one edge, e = (a,b), fixed. The next theorem tells us that this suffices to solve the isomorphism problem.

Theorem 6.1: Trivalent graph isomorphism testing is polynomial-time reducible to computing the automorphism group of a trivalent graph with one edge fixed.

Proof : Let G and H be two connected trivalent graphs. Pick any edge e in G and add a new vertex v to G on this edge. Pick any edge e' in H and add a new vertex v' to H on this edge. Call these two graphs G' and H' respectively. Construct a new graph X that is G' + H' together with a new edge between v and v'. Compute the automorphism group of X that fixes (v,v'). If there is a generator that interchanges G' and H' then G and H are isomorphic. By trying this for every possible choice of edge e' in H we can determine if G and H are isomorphic.

□

6.1 The Algorithm

Take the vertices of G and label them according to their distance from the edge e. Let the group A_i be the automorphisms of G_i, the subgraph consisting of vertices and edges out to distance i from e. The graph G_0 is just the single edge (a,b), and A_0 is the group of order two consisting of the identity and the permutation that interchanges a and b.

It is our intention to compute A_{i+1} from A_i.

A_i is a 2 group. Let $\overline{A}_i$ be its action on the level i vertices, also a 2 group. Construct $Syl(\overline{A}_i)$, a 2SSG, of the symmetric group on level i vertices that contains $\overline{A}_i$.

Form a cone graph from the tree representation of $Syl(\overline{A}_i)$ by taking each vertex of level i+1 and creating a hyperedge out of the set of level i vertices it is adjacent to. Using the modified version of Hoffmann's algorithm, calculate H_i, the automorphism group of this cone graph. Let $\overline{H}_i$ be H_i restricted to the level i vertices.

The group $\overline{H}_i \cap \overline{A}_i$ consists of just those permutations of level i vertices that can be extended to automorphisms of G_{i+1}. Generators for $\overline{H}_i \cap \overline{A}_i$ can be determined in polynomial time since both groups lie under $Syl(\overline{A}_i)$. The automorphism group A_{i+1} will be constructed by extending $X = \overline{H}_i \cap \overline{A}_i$ to act on G_{i+1}.

To get at A_{i+1} we first construct B_i, the subgroup of A_i whose restriction to level i vertices is in X. To test whether an element y is in B_i, we can restrict y to operate on the level i vertices and test if the restriction is in X. Therefore, B_i is polynomial-time recognizable.

When X was constructed, a polynomially accessible tower from $\bar{A}_i$ through X to I could have been found. Let $\bar{A}_i \supset D_1 \supset \cdots \supset D_t = X \supset \cdots \supset I$ be this tower. Define R_j to be the subgroup of A_i which, when restricted to level i vertices, is in D_j.

The group B_i can be trapped in the polynomially accessible tower $A_i \supset R_1 \supset \cdots \supset R_t = B_i \supset C_1 \supset \cdots \supset I$, in which the C_j are just groups fixing successively more vertices. From this tower, generators $b_1, \ldots, b_r$ for B_i can be found in polynomial time. Having these generators, B_i can be extended to A_{i+1}.

Let F_{i+1} be the subgroup of A_{i+1} in which level i vertices and below are fixed. Find generators for F_{i+1} by direct examination of the graph G_{i+1}. For each b_j, consider its action on level i vertices and pick any extension $\bar{b}_j$ that includes level i+1 vertices and is an automorphism of G_{i+1}. Generators for A_{i+1} are the $\bar{b}_j$ together with generators for F_{i+1}.

6.2 Timing Analysis

The only step that takes more than a polynomial amount of time is determining the automorphism group of a cone graph. The number of times this step is performed corresponds to the number of levels in the graph, which is at most n. Therefore, the whole algorithm runs in time $O(n^{c \log n})$.

References

[1] A. Aho, J. Hopcroft, J. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley (1974).

[2] L. Babai, "Isomorphism Testing and Symmetry of Graphs," unpublished manuscript.

[3] L. Babai, "Monte-Carlo Algorithms in Graph Isomorphism Testing," submitted to SIAM J. on Computing (1979).

[4] K. Booth, C. Colbourn, "Problems Polynomially Equivalent to Graph Isomorphism," Tech. Report CS-77-04, Computer Science, Univ. Waterloo (1977).

[5] C. Colbourn, "A Bibliography of the Graph Isomorphism Problem," Tech. Report 123/78, Computer Science, Univ. Toronto (1978).

[6] R. Dean, Elements of Abstract Algebra, John Wiley and Sons (1966), p. 243.

[7] M. Furst, J. Hopcroft, E. Luks, "A subexponential Algorithm for Trivalent Graph Isomorphism," Proc. Eleventh Southeastern Conf. on Graph Theory and Computing (1980), to appear.

[8] M. Hall, The Theory of Groups, Macmillan (1959).

[9] C. Hoffmann, "Testing Isomorphisms of Cone Graphs," Proc. Twelfth Annual ACM Symposium on the Theory of Computing (1980), to appear.

[10] R. Mathon, "A Note on the Graph Isomorphism Counting Problem," Information Processing Letters 8, pp. 131-132.

[11] R. Read, D. Corneil, "The Graph Isomorphism Disease," J. Graph Theory 1 (1977), pp. 339-363.

[12] C. Sims, Computational Problems in Abstract Algebra, John Leech, ed., Pergamon Press (1970), pp. 176-177.