CIS-TR-80-11

A LINEAR ALGORITHM TO DETERMINE ISOMORPHSIM OF UNICYCLIC GRAPHS

by

Sandra Mitchell Hedetniemi *
Terry Beyer

Department of Computer and Information Science
University of Oregon
Eugene, OR 97403

* Research sponsored in part by the National Science Foundation under Grant MCS-79-03913

ABSTRACT

A linear algorithm is presented for determining the isomorphism of two unicyclic graphs (graphs which are connected and have exactly one cycle). This algorithm is simpler than the general algorithm for planar graphs [9] which could also be applied since it uses the properties of the class of unicyclic graphs. It therefore is easily generalized to the class of functional digraphs. The algorithm uses as a subalgorithm a linear algorithm to uniquely encode a tree as a sequence of integers.

CR Categories 5.32
Keywords: graph isomorphism, tree encoding, unicyclic graphs, functional digraphs

1. Introduction

Two graphs $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ are isomorphic if there exists a 1 - 1 function $f: V_1 \rightarrow V_2$ from V_1 onto V_2 such that if $(u_1,v_1) \in E_1$, then $(f(u_1),f(v_1)) \in E_2$. The graph isomorphism problem has been studied in the literature [5] and [12]. Although it is not known whether the general isomorphism problem is NP-complete, it has been shown that the related subgraph isomorphism problem is NP-complete [7]. One of the better known algorithms, due to Corneil and Gotlieb [6], has an exponential time complexity in the worst case.

The general isomorphism problem has been shown by Booth and Leuker [4] to be polynomial reducible to the isomorphism problem for chordal graphs. Linear time solutions exist for two special classes of chordal graphs: trees (Hopcroft and Tarjan [8]) and maximal outerplanar graphs (Beyer, Jones and Mitchell [2]).

A linear algorithm also exists for the class of planar graphs (Hopcroft and Wong [9]). In this paper, a simple linear algorithm is presented for the class of unicyclic graphs, a subclass of planar graphs. This solution uses a subalgotithm which finds a canonical encoding for a rooted tree in linear time.

2. Unicyclic graphs

A <u>unicyclic graph</u> G = (V,E) is a connected graph with exactly one cycle C; the vertices of this cycle $C = \{v_1,v_2, \ldots, v_k\}$ are called <u>cycle vertices</u>. The <u>branch of cycle vertex v</u>, denoted T_v, is the subtree rooted at v containing no other cycle vertices than v.

The isomorphism problem on two unicyclic graphs is equivalent to finding a 1-1 mapping between their respective sets of branches which preserve adjacencies between the cycle vertices and under which corresponding branches are isomorphic.

The equivalent problem immediately suggests application of an existing linear algorithm for trees by Hopcroft and Tarjan [8]. This algorithm requires as input two rooted, unlabeled trees and answers the question of isomorphism by processing the trees by levels of vertices beginning at the level farthest from the root. The level l_i of vertex i is defined to be the distance of vertex i from the root plus 1, where the root is considered to be at distance 0 from itself. Unfortunately, Hopcroft and Tarjan's algorithm requires intermediate verification of isomorphism after processing each level in the trees. This process would be an $O(N^2)$ process for two unicyclic graphs with N vertices since it is equivalent to comparing each branch in one graph to every branch in the other.

A more suitable approach for determining the 1-1 mapping between isomorphic branches is to associate with each branch a canonical sequence of integers such that branches have identical sequences if and only if they are isomorphic. In the following section we present a linear time algorithm to construct an integer "level" sequence for the branches. This sequence has the desired canonical property.

3. Linear time coding of rooted trees

We associate with each ordered rooted tree T' a level sequence, L(T'), which is obtained by traversing T in preorder (cf. Knuth [10]) and recording the level of each vertex as it is visited (cf. Figure 1). We claim without proof that any two ordered, rooted trees having the same level sequence are isomorphic.

A given unordered, rooted tree T may correspond to many non-isomorphic, ordered trees and hence, might be represented by any one of the corresponding level sequences. The canonical representation of T is that unique corresponding ordered tree, T', whose level sequence, L(T'), is greater than the others in the

lexicographic ordering of integer sequences. The canonical level sequence of T is $L(T)^* = L(T')$.

In Figure 1 the two ordered trees T' and T" correspond to the same underlying rooted tree T (not shown). Since these are the only distinct ordered trees corresponding to T and since $L(T') > L(T'')$, we have that T' is the canonical representative of T and the canonical level sequence of T is $L(T)^* = (1\ 2\ 3\ 3\ 2)$.

$L(T'') = (12233)$ $\qquad$ $L(T') = (]2332)$

$L(T)^* = L(T')$

Figure 1

The canonical represenative of a rooted tree has a distinguishing property which we now define. Let T' be an ordered tree. Let T_i and T_j be subtrees rooted at vertices i and j respectively. We say T_i is the <u>immediate predecessor</u> of T_j if i and j are consecutive children of the same parent with i to the left of j. An ordered tree is <u>regular</u> if $L(T_i) \geq L(T_j)$ whenever T_i is the immediate predecessor of T_j.

The proof of the following result can be found in [2].

<u>Lemma 1</u>. An ordered tree is the canonical represenative of its underlying rooted tree if and only if it is regular.

Lemma 2. A subtree of a regular tree is regular.

Proof. This follows immediately from the transitivity of the relation subtree and the definition of regular.

To solve our problem we need to create an ordering on the rooted tree which is regular. We will do this, by virtue of Lemma 2, by creating ordered subtrees, working upward from the deepest levels. Each subtree will be ordered based on the level sequence representing the vertices in that subtree.

We next present Algorithm ENCODE which produces a canonical level sequence of a given unordered rooted tree. This algorithm uses a linear list of elements which are of the following types:

(i) hard boundaries (denoted ▮ in Figure 2),

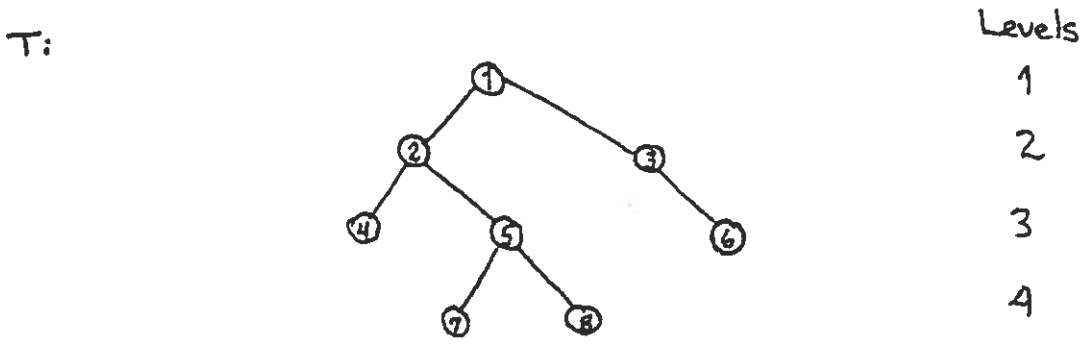
(ii) soft boundaries (denoted ▯ in Figure 2), and

(iii) subtree descriptors consisting of a vertex v and a subsequence S of the canonical level sequence (denoted v(S) in Figure 2).

The boundary elements (hard and soft) serve to partition the subtree descriptors into blocks according to their subsequences. Boundaries are created as soft boundaries as a result of processing some other block B (except during initialization). Upon completion of the processing of block B, the newly created soft boundaries become hard.

The initialization of the list of elements for a tree T involves the following steps. Associate with each vertex v it's subtree descriptor S_v consisting solely of v's level in the tree. Create the list by inserting depth(T)+1, i.e. the maximum number of levels plus one, soft boundaries. Insert each vertex v, having level k, between the k-th and the (k+1)-st soft boundary.

Processing is performed on the list of elements from right to left. Let B, the current block being processed, contain vertices $v_1, v_2, \ldots, v_j$ with their respective subsequences $S_{v_1}, S_{v_2}, \ldots, S_{v_j}$. Let $p_1, p_2, \ldots, p_j$ be the



Algorithm Step	State of Sequence			
	Level 1	Level 2	Level 3	Level 4
1	1(1)	2(2) 3(2)	4(3) 5(3) 6(3)	7(4) 8(4)
2.2	1(1)	2(2) 3(2)	4(3) 6(3) 5(34)	7(4)
2.2	1(1)	2(2) 3(2)	4(3) 6(3) 5(344)	
7.8	1(1)	2(2) 3(2)	4(3) 6(3) 5(344)	
2.2	1(1)	3(2) 2(2344)	4(3) 6(3)	
2.8,2.8	1(1)	3(2) 2(2344)	4(3) 6(3)	
2.2	1(1)	3(23) 2(2344)	4(3)	
2.2	1(1)	3(23) 2(23443)		
2.8	1(1)	3(23) 2(23443)		
2.2	1(123443)	3(23)		
2.8,2.8	1(123443)	3(23)		
2.2	1(12344323)			
2.8,2.8	1(12344323)			

3 $L(T)^* = (12344323)$

Figure 2. An application of Algorithm ENCODE

parents of v_1, v_2, ..., v_j, repsectively. Then for each i, $1 \leq i \leq j$, processing of vertex v_i causes:

(1) p_i's subsequence to be updated by concatenating S_{v_i} to its end;

(2) p_i to be placed in a different block (to maintain the equivalence partition), with the possible creation of a soft boundary (if a new block is required);

(3) vertex v_i to be removed from the list of elements.

Figure 2 contains an example of the processing of Algorithm ENCODE on a tree with four levels.

Algorithm ENCODE. To find the canonical level sequence $L(T)^*$ for an unordered, rooted tree T having N vertices and M levels, using a list of elements which are either boundaries (hard and soft) or subtree descriptors.

1. [Initialize the list of elements]
   1.1. Construct a sequence of M+1 soft boundaries.
   1.2. For each vertex v in T, construct a subtree descriptor v(k), where k is the level number of v in T. Insert this descriptor between the k-th and (k+1)-st boundary.
2. [Process the list of elements]
   2.1. While the rightmost element is not the descriptor for the root do
      2.2. while the rightmost element is a descriptor do
         2.3. let v(s) be the rightmost descriptor
         2.4. let p(t) be the descriptor such that p is the parent of v
         2.5. let b the first boundary (hard or soft) to the right of p(t)
         2.6. remove p(t) from the sequence
         2.7. concatenate s to t, forming p(ts)

2.8. if b is a soft boundary

2.9. then insert p(ts) immediately to the right of b

2.10. else [b is a hard boundary]

2.11. insert a new soft boundary immediately to the left of b

2.12. insert p(ts) immediately to the left of b

fi

2.13. remove v from the list of elements

od.

2.14. while the rightmost element is a boundary do

2.15. remove the rightmost element

2.16. convert all soft boundaries in the sequence to hard boundaries

od.

od.

3. [Recover results]

3.1. let the rightmost element be r(u)

3.2. r is the root and $L(T)^* = (u)$.

Lemma 3. Let $u(S_u)$ and $v(S_v)$ be subtree descriptors in the list of elements with $u(S_u)$ to the left of $v(S_v)$. Then the following are true:

(1) if at least 1 boundary exists between $u(S_u)$ and $v(S_v)$, then $S_u < S_v$; and

(2) if no boundary exists between $u(S_u)$ and $v(S_v)$, then $S_u = S_v$.

Proof. Properties (1) and (2) are initially true by the definition of the partition. We now prove the properties are invariant during the processing of the algorithm.

Steps 2.13 and 2.15 do not affect these properties since they only remove

the rightmost element. Furthermore, since we do not distinguish between hard and soft boundaries, Step 2.16 does not affect these properties (directly). It only remains to show the effect of Steps 2.6 through 2.12 since Steps 2.3 through 2.5 are designational. Therefore assume the rightmost elemet w is a descriptor and its parent is one of u or v.

Case 1. $S_u = S_v$, and without loss of generality, the parent of w is u. The concatenation of S_w to S_u, i.e. $S_u' = S_u S_w$ creates the inequality $S_u' > S_v$. But either Step 2.9 or Steps 2.11 and 2.12 will cause S_u' to be placed to the right of S_v with a boundary between them.

Case 2. $S_u < S_v$ and the parent of w is u.

Let $S_u' = S_u S_w$; $S_u = s_{u1}\, s_{u2} \cdots s_{uj}$; $S_v = s_{v1}\, s_{v2} \cdots s_{vk}$; and i be the rightmost position such that $s_{um} = s_{vm}$ for all $m < i$. If $i \leq j$, then S_u and S_v will have a hard boundary between them since they cannot have become distinguishable during the processing of the block in which w resides. Hence, $S_u' < S_v$ and the hard boundary will remain between them.

If $i = j+1$, i.e. S_u is a proper initial segment of S_v, then two possibilities exist: either S_w is equal to $s_{v(i+1)}\, s_{v(i+2)} \cdots s_{vk}$ or it is not. If they are equal, then there exists exactly one soft boundary between $u(S_u)$ and $v(S_v)$. There cannot be more than one soft boundary since that would imply $s_{v(i+1)} \cdots s_{vk}$ was not appended to $s_{v1} \cdots s_{vi}$ as a "unit" subsequence. This would violate Property (1) since S_w would be in the same block as a "lesser" subtree descriptor. There cannot be one or more hard boundaries since Property (2) would be violated by having S_w to the left of a "lesser" subtree descriptor.

If they are not equal, then S_w must be less than the designated subsequence. Otherwise, S_w is greater than this subsequence and earlier in the processing either there was one boundary between the two subsequences, violating Property(1), or there was none, violating Property (2).

Case 3. $S_u < S_v$ and the parent of w is v.

Clearly $S_u < S_v' = S_v S_w$. Since the algorithm only moves elements further to the right, Property (1) is preserved.

Hence, in all cases, Properties (1) and (2) have been shown to be invariant with respect to the algorithm. ∎

Theorem 4. Let $v(S_v)$ be a subtree descriptor in the list of elements where $v(S_v)$ lies in the rightmost, non-empty block. Let S_v^* denote the sequence obtained by subtracting (1 - level of v in T) from each element in S_v. Then the subtree rooted at v is canonically represented by S_v^*.

Proof. Initially, this is true since the rightmost block contains only end-vertices. Lemma 3 guarantees that the property of dominance between subtree descriptors is initially true and invariant during the operation of the algorithm. When a vertex lies in the rightmost non-empty block, its subtree is completely processed. Because the blocks are processed right-to-left, that vertex's descriptor is regular. Subtracting a constant from every element in the descriptor does not change the property of being regular (the level of the vertex is only raised). By Lemma 1, this regular representation is therefore a canonical representation. ∎

Corollary 5. Algorithm ENCODE finds a canonical representation of a tree.

Proof. When Algorithm ENCODE terminates, only one vertex remains in the tree, the root. By Theorem 4, the sequence obtained by subtracting 0 from the subtree descriptor is a canonical representation of the subtree rooted at the root of T, i.e. the entire tree. ∎

The linearity of Algorithm ENCODE is demonstrated by the following. Initially, the list of elements contains N subtree descriptors and $M \leq N+1$ soft boundaries, The processing of each subtree descriptor creates no more

than one soft boundary. Hence, the total number of iterations of the loop in Step 2.1 is $\leq$ 3N+1. It only remains to show that each of the steps can be performed in constant time.

The tree will be stored as a parent array (cf. Knuth [10]). The list will be maintained as a doubly linked list where each subtree descriptor has an additional pointer to the nearest boundary to the right. Hence, all accesses are made in constant time. However, it is important that each subtree descriptor always have a pointer to the correct nearest right boundary. Assume u and v_1, v_2, ..., v_k are in the same block, their nearest right boundary b is a hard boundary, u is the parent of w, and w is the rightmost element. Execution of Step 2.11 will cause a new soft boundary to intervene between the v_i's and b. If the algorithm is to execute in linear time, it is essential that all k right boundary pointers not be updated. A simple trick solves this problem. Rewrite Steps 2.11 and 2.12 as follows:

2.10. else [b is a hard boundary]
  2.11 insert a new soft boundary immediated to the right of b
  2.12a insert p(ts) immediately to the right of b
  2.12b change the new boundary to be hard and make b a soft boundary.

This insures that all updates can be made in constant time as well, guaranteeing the linearity of the algorithm.

4. Isomorphism of unicyclic graphs

Algorithm ENCODE can be used to uniquely encode the branch of each cycle vertex v in a unicyclic graph. These level sequences, each beginning with a 1, can be concatenated in a cyclic fashion to represent the unicyclic graph canonically upto clockwise and counterclockwise permutations of the cycle.

The next result follows immediately from the canonical representations

of the branches.

Theorem 6. Let G_1 and G_2 be unicyclic graphs and let $L_C(G_1)$ be the string obtained by concatenating the level sequences of the branches of G_1 in clockwise order and let $L_{CC}(G_1)$ be the string obtained using counterclockwise order, where the first branch has been chosen arbitrarily. Let $L_C(G_2)$ be described equivalently for G_2. Then G_1 is isomorphic to G_2 if and only if $L_C(G_1)$ is a subsequence of $L_C(G_2)L_C(G_2)$ or $L_{CC}(G_1)$ is a subsequence of $L_C(G_2)L_C(G_2)$.

The linearity of the algorithm to determine isomorphism of unicyclic graphs is therefore dependent upon the linearity of Algorithm ENCODE and the efficiency of executing the proof of Theorem 6. But Morris and Pratt's [11] substring pattern matching algorithm which is presented in [1] is linear in the length of the two strings. In this case this length is linear in the number of vertices in the unicyclic graphs.

5. Concluding remarks

Although the algorithm is presented in this paper does not actually solve isomorphism for a new class of graphs, it greatly simplifies the more general planar algorithm that previously would have been used for unicyclic graphs. It should be noted that recognition of unicyclic graphs is also a linear process. The algorithm presented in this paper relies upon a very useful algorithm to uniquely encode a tree in linear time. This has particular relevance to the encoding of tree-like chemical compounds. Furthermore, it provides a mechanism for determining isomorphsim of directed trees and hence functional digraphs.

6. Bibliography

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

[2] Beyer, T. and Hedetniemi, S. M., Constant time generation of trees, SIAM J. Comput., to appear.

[3] Beyer, T., Jones, W. and Mitchell, S., Linear algorithms for isomorphism of maximal outerplanar graphs, J. Assoc. Comput. Mach., 26(1979), 603-610.

[4] Booth, K. and Leuker, G., Linear algorithms to recognize interval graphs and test for consecutive ones property, J. Comput. and Sys. Sci., 13(1976), 335-379.

[5] Colburn, C., A bibliography of the graph isomorphism problem, Univ. of Toronto Tech. Rept. 123/78, 1978.

[6] Corneil, D. and Gotlieb, C., An efficient algorithm for graph isomorphism, J. Assoc. Comput. Mach., 17(1970), 51-64.

[7] Garey, M. and Johnson, S., Computers and Intractability: a guide to the theory of NP-completeness, (W. H. Freeman and Co.: San Francisco), 1979.

[8] Hopcroft, J. and Tarjan, R., Isomorphism of planar graphs, in Complexity of Computations (R. Miller and J. Thatcher, eds.) (Plenum Press: New York), 1972, 143-150.

[9] Hopcroft, J. and Wong, J., Linear time algorithm for isomorphsim of planar graphs, Proc. Sixth ACM Symposium on Theory of Computing, 1974, 172-184.

[10] Knuth, D., Fundamental Algorithms, (Addison-Wesley: Reading, Mass.), 1969.

[11] Morris, J. and Pratt, V., A linear pattern matching algorithm, Univ. of California Tech. Rept. 40, Computing Center, University of California, Berkeley, 1970.

[12] Read, R. and Corneil, D., The graph isomorphism disease, J. Graph Theory, 1(1977), 239-263.