Linear Time Algorithm for Isomorphism of Planar Graphs[†]

(Preliminary Report)

J.E. Hopcroft and J.K. Wong
Department of Computer Science
Cornell University
Ithaca, N. Y. 14850

The isomorphism problem for graphs G_1 and G_2 is to determine if there exists a one-to-one mapping of the vertices of G_1 onto the vertices of G_2 such that two vertices of G_1 are adjacent if and only if their images in G_2 are adjacent. In addition to determining the existence of such an isomorphism, it is useful to be able to produce an isomorphism-inducing mapping in the case where one exists.

The isomorphism problem for triconnected planar graphs is particularly simple since a triconnected planar graph has a unique embedding on a sphere [6]. Weinberg [5] exploited this fact in developing an algorithm for testing isomorphism of triconnected planar graphs in $O(|V|^2)$ time where V is the set consisting of the vertices of both graphs. The result has been extended to arbitrary planar graphs and improved to $O(|V|\log|V|)$ steps by Hopcroft and Tarjan [2,3]. In this paper, the time bound for planar graph isomorphism is improved to $O(|V|)$. In addition to determining the isomorphism of two planar graphs, the algorithm can be easily extended to partition a set of planar graphs into equivalence classes of isomorphic graphs in time linear in the total number of vertices in all graphs in the set. A random access model of computation (see Cook [1]) is assumed. Although the proposed algorithm has a linear asymptotic growth rate, at the present stage of development it appears to be inefficient on account of a rather large constant. This paper is intended only to establish the existence of a linear algorithm which subsequent work might make truly efficient.

1. The Algorithm for Planar Graph Isomorphism

Previous work on isomorphism of planar graphs cited above shows that without loss of generality we can restrict attention to determining isomorphism of embeddings of triconnected planar graphs. If G_1 and G_2 are nontrivial, triply connected planar graphs, they have a unique representation on a sphere only up to parity (that is, left or right depend on whether the graph is viewed from inside or outside the sphere). Thus one must actually test isomorphism of one planar representation of G_1 with both representations of G_2 in order to determine if G_1 and G_2 are isomorphic. Henceforth in this paper we restrict our attention to the isomorphism of fixed embeddings of planar graphs. From now on the word <u>graph</u> refers to a specific labelled planar representation of a planar graph.

---

† This research was supported by the Office of Naval Research under grant number N 00014-67-A-0077-0021. Reproduction in whole or in part is permitted for any purpose of the United States Government.

The algorithm associates integer labels with vertices and pairs of integer labels with edges, one label with each end. The integer associated with a vertex is called the vertex label. Let edge e be incident at vertices u and v. The integer associated with the vertex u end of e is called the u-label of e and the integer associated with the vertex v end of e is called the v-label of e.

Next each graph is simplified by a sequence of reductions. A reduction of graph G is a replacement of each labelled subgraph of G of a given type by a labelled subgraph of another given type. A list of possible reductions, each having an associated priority, is detailed later.

The isomorphism algorithm assigns the label 1 to each vertex and the label 2 to each edge end. Then the highest priority reduction which is applicable is applied to G_1 and G_2. Certain discrepancies may be detected at this stage in which case the algorithm terminates and G_1 and G_2 are not isomorphic. For example, if the number of subgraphs of the type to be collapsed by the reduction differ in G_1 and G_2, then clearly the graphs are not isomorphic. The process of applying reductions continues until no further reduction is applicable. At every stage the highest priority applicable reduction is applied.

The crucial property of the reductions is that their result must be as if all similar subgraphs were acted upon simultaneously in an identical manner. Since the actual process of applying a reduction sequentially collapses all subgraphs of a given type, the subgraph modifications cannot interfere with each other if the result is to be order independent. A second equally important property of reductions is that the modified labels encode sufficient information to insure that the resulting graphs are isomorphic if and only if the original graphs are isomorphic. (To ensure that both graphs receive the same labels in each reduction, label assignments for each reduction are always done simultaneously for both graphs.)

Reductions are performed in the order determined by their priority. This priority ordering insures a canonical form for the graph at each stage. After each reduction the graph is simplified in the sense that there is a strict decrease in the complexity of the graph as measured by the sum of the number of edges and vertices.

The work done to achieve this decrease in the sum of the number of edges and vertices is proportional to the decrease. The fact that a triconnected planar graph has a number of edges less than three times the number of vertices insures termination of the algorithm in time which is linear in the number of vertices.

When no further reduction is applicable, the graphs that can be present are the five regular polyhedral graphs or a trivial graph consisting of a single vertex. These graphs can be tested for isomorphism (as labelled graphs) by exhaustive matching in a fixed finite time.[†]

To clarify the process consider the reduction shown in Figure 1. The graph of Figure 1a has two square faces, each of which has been collapsed to a single vertex in Figure 1b. The vertex is labelled to indicate that it originally was a square face. The next collapse would replace the bonds by single edges labelled to indicate that they represent a bond and to distinguish them from other edges which are edges of the original graph. This reduction would reduce the graph to a regular octahedron. Observe that we cannot readily collapse the triangular faces in Figure 1a since the collapse of one

† Further reductions may also be defined for the regular polyhedral graphs if all the labels on such a graph are not the same.

triangular face will interfere with the collapse of an adjacent one.

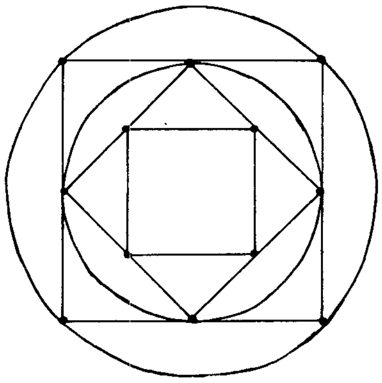
Figure 1a

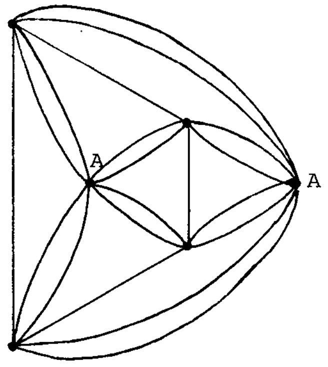

Figure 1b

The highest priority reductions involve removal of loops, bonds and degree one vertices. Once loops, bonds and degree one vertices have been removed, Euler's theorem relating edges, faces and vertices in a planar graph (Ore, [4]) guarantees the existence of vertices of degree d where d equals 2, 3, 4 or 5. Subsequent reductions involve removal of these low degree vertices.

2. Two Subalgorithms

Two important subalgorithms are used to encode information into labels.

A. vector partitioning

The vector partitioning algorithm partitions a set of varying length n-tuples of integers so that $(a_1,\ldots,a_m)$ and $(b_1,\ldots,b_n)$ are placed in the same equivalence class iff

$$(1)\quad m = n \quad\text{and}\quad (2)\quad a_i = b_i\ ,\quad 1 \le i \le n\ .$$

The method, utilizing a pigeonholing technique, requires time proportional to the sum of the lengths of the n-tuples. Initially the set of n-tuples to be partitioned is in one equivalence class.

```
vector partition algorithm
   while an n-tuple with an unpigeonholed component exists do
   begin
      for each class with an unpigeonholed component do
      begin
         for each n-tuple in this class do
            pigeonhole the n-tuple† on the first unpigeonholed component if such a
               component exists, otherwise place into a special pigeonhole, putting
               location of each new pigeonhole used on a stack;
         while stack of occupied pigeonholes $\ne \phi$ do
         begin
            empty the n-tuples placed into the pigeonhole referenced by the top
               entry on the stack into a new equivalence class;
            pop top entry from stack
         end
      end
   end
```

† Actually a pointer to the n-tuple rather than the n-tuple itself is placed in the pigeonhole.

The algorithm is used to "encode" n-tuples of integers into a single integer to be used for relabelling. A function NEW_INTEGER giving the next unused integer is used to assign integer "names" to each n-tuple equivalence class. NEW_INTEGER increments itself automatically on each invocation. If $(a_1,\ldots,a_m)$ is placed in the equivalence class "named" n by partitioning P, we write $\mathrm{ENCODE}_P(a_1,\ldots,a_m)$ for n and DECODE(n) for $(a_1,\ldots,a_m)$. The execution time of the n-tuple partitioning algorithm is easily seen to be proportional to the sum of the lengths of the initial n-tuples.

B. circle partitioning

Consider a second equivalence relation between varying length n-tuples of integers such that $(a_0,\ldots,a_{m-1})$ is equivalent to $(b_0,\ldots,b_{n-1})$ iff

(1) $m = n$ (2) for some fixed k
$a_i = b_{(i+k)\bmod n}$, $0 \le i \le n-1$.

An n-tuple under this equivalence relation is called an n-circle. In an n-circle $(a_0,\ldots,a_{n-1})$, $a_{(i+1)\bmod n}$ is said to be counterclockwise adjacent to a_i, $0 \le i \le n-1$. a_i is clockwise adjacent to a_j iff a_j is counterclockwise adjacent to a_i.

```
circle isomorphism algorithm
   while there exist two distinct integers in some circle do
   begin
      for each occurrence of an ordered pair a,b where a is clockwise adjacent to b
         in some circle and $a \neq b$ do
         construct a 2-tuple (a,b) associated with this occurrence;†
      partition this set of 2-tuples using the n-tuple partitioning algorithm,
         naming each equivalence class from NEW_INTEGER;
      call this partitioning P;
      for all classes in order of their names do
         for each 2-tuple (a,b) in the class do
         begin
            replace the occurrence of the two components, (if they both still exist),
               associated with (a,b) by a single component whose value is $\mathrm{ENCODE}_P(a,b)$;
            comment: this results in shrinking the circle containing the occurrence
               by one for each such replacement;
         end
   end
   if both circles' dimensions are not the same or the value common to all components
      of one circle is not the same as that common to all components of the other then
      stop with no isomorphism otherwise stop with isomorphism
```

The above algorithm decides isomorphism of two n-circles in O(n) steps. A simple modification will partition a set of circles into classes of "isomorphic" circles in time proportional to the sum of the lengths of the circles in the initial set. Analogous to the n-tuple case, we can use the above algorithm to "encode" n-circles into a single integer.

3. The Reductions

A. Removal of loops and degree one vertices

The highest priority reductions involve loops, degree one vertices and bonds. We list them here in order of priority.

A loop is an edge such that both ends are incident at the same vertex v and whose

† For each subsequent iteration of the while loop these pairs (a,b) can be found and constructed in time proportional to the number of pairs of the previous iteration.

edge ends are adjacent. The vertex v is called a loop vertex. If v has no non-loop edges and the number of loops is greater than one, the graph consisting of vertex v and the edges incident at v is called a corolla. If v has no non-loop edges and only one loop, the graph is called a knot.

Suppose e_1 and e_2 are incident at w and that e_1 immediately precedes e_2 in the clockwise ordering of edges at w in the planar embedding. Then we write $e_2 CW_w e_1$. We write $e_1 CCW_w e_2$ to denote $e_2 CW_w e_1$.

(i) Removing loops from loop vertices with non-loop edges

A loop tuple is a triple (non-loop edge e = (v,w), loop edge f, loop vertex v) such that edge e is counterclockwise adjacent to the loop f at vertex v. The number triple of a loop tuple (e,f,v) is the ordered triple (v-label(e),† label of end of f clockwise adjacent to edge e, other label of f). The reduction consists of constructing the corresponding number triple for each loop tuple, partitioning the set of number triples using the vector partitioning algorithm, assigning each number triple an integer based on its equivalence class, assigning the integer associated with loop tuple (e,f,v) to the v-label of e, and removing the loop f.

Given a list of loop tuples for a graph G, the above reduction produces a unique resultant graph G' independent of the order of the list of loop tuples. Furthermore, the algorithm can be implemented in time linear in the number of loops removed. In our running time analyses, from now on we assume the subgraphs on which subsequent reductions are to be performed have already been detected and do not enter into the cost of a reduction.

Observe that by recording the number triple assigned a given integer by the vector partitioning algorithm, the graph G can be recovered from G'.

(ii) Removing corollas

Let c be a corolla with loop vertex v. For each loop ℓ of c with ends $\hat{e}$ and $\hat{f}$ let $\hat{e}$ be the end of ℓ which is clockwise adjacent to the $\hat{f}$ end of ℓ. Assign the vector (label(v), $\hat{e}$-label(ℓ), $\hat{f}$-label(ℓ)) to the loop ℓ. For each corolla c, form a circle of 3-tuples with the vectors assigned to the loops of c as components in the obvious manner. Partition the circles (ignoring the 3-tupling) using the circle partitioning algorithm and assign integers to the circles based on the partitioning. The loop vertex of each corolla is assigned the integer associated with the corresponding circle. All loops of the corolla are then removed.

(iii) Removing knots

A knot with loop vertex v and loop ℓ is represented by a circle (label(v), first label of ℓ, second label of ℓ, label(v), second label of ℓ, first label of ℓ). The reduction is then analogous to that for corollas. The only reason for treating knots separate from corollas is the inability to distinguish between the two ends of a loop in a knot.

(iv) Removing degree one vertices

A spoke is a degree one vertex and its associated edge. Each degree one vertex is

† The usage is ambiguous for e = (v,v) but the meaning should be clear.

associated with a unique spoke. A <u>spoke center</u> is a vertex incident to a spoke. If the spoke center has no edges incident other than spoke edges and the number of spoke edges is greater than one, then we have a <u>star</u>. If we have just one spoke we have a <u>dumbell</u>. Notice that the spoke of a dumbell is not unique.

Spoke centers with non-spoke edges incident are handled in an analogous manner to loop vertices with non-loop edges, stars are handled in an analogous manner to the handling of corollas, and dumbells are handled analogous to knots.

B. Bond associated reductions

A <u>clump</u> is a maximal set of edges $e_1,\ldots,e_k$, $k > 1$, connecting two distinct vertices v and w such that

(1) at least one of v and w is adjacent to a vertex other than v and w,

(2) $e_i CCW_v e_{i+1}$ and $e_i CW_w e_{i+1}$ for $1 \leq i < k$.

The two vertices v and w are called <u>clump vertices</u>. The <u>clump vector</u> of v is the vector (v-label(e_1),...,v-label(e_k)) and the clump vector of w is the vector (w-label(e_k),..., w-label (e_1)).

For each clump we create the two clump vectors, partition the set of all clump vectors using the vector partitioning, and assign integers to the clump vectors. Each clump is replaced by a single edge labelled with the integers associated with the two appropriate clump vectors.

A <u>skein</u> is a graph consisting of two vertices u and w and k edges, $k > 1$, each edge incident at both u and w. The vertices u and w are the <u>skein vertices</u> and the k edges are the <u>skein edges</u>.

For each skein with skein vertices v and w and skein edge f associate the vector (label(v), v-label(f), w-label(f), label(w)) with the v end of f and (label(w), w-label(f), v-label(f), label(v)) with the w end of f. Then form a circle of 4-tuples around each skein vertex in obvious manner and partition the circles by the circle partitioning algorithm. In this manner each vertex in a skein is associated with an integer. Replace each skein by a vertex labelled with the smaller of the two integers.

C. Four general classes of reductions and two special cases

Once loops, bonds and degree one vertices have been removed, Euler's theorem guarantees the existence of a degree 2, 3, 4 or 5 vertex. With this in mind, we call a vertex of degree 2, 3, 4 or 5 a <u>low degree vertex</u>. Thus we need only insure that we can apply a reduction whenever a low degree vertex exists.

The remaining reductions, in order of priority, are as follows. The first general class of reductions is the replacement of all degree d vertices, all of whose neighbors are of degree other than d, by a d-gon. This is done for d = 2, 3, 4 and 5. At this point either the graph is a regular degree d graph or there exists a degree d vertex which is adjacent to a non-degree d vertex, d = 2, 3, 4 or 5. The next class of reductions collapses an edge connecting a degree d vertex with a non-degree d vertex. This also is done for d = 2, 3, 4 and 5. The final two general classes of reductions handle graphs which are regular degree d. There are also two special reductions, involving degree four vertices.

(i) Removing isolated low degree vertices

A degree d vertex which is not directly connected to another degree d vertex is said to be isolated. Isolated degree d vertices are removed and replaced by d-gons. Specifically for each isolated vertex v we do the following. For each pair of adjacent edges (v,u) and (v,w) incident at v with (v,u)CCW_v(v,w), edge (u,w) is added such that (u,w)CCW_u(u,v) and (v,w)CCW_w(u,w). Observe that the edges (u,w), (u,v) and (v,w) form a triangular face in the planar embedding. Associate vector (label(v), v-label(u,v), u-label(u,v)) with the u end of (u,w) and associate the vector (w-label(w,v), v-label(w,v), label(v)) with the w end of (u,w). Partition the class of all vectors created, assign integers to the vectors and label the appropriate edge ends with these integers. Finally remove v and all edges incident with v.

Observe that the above reduction removes all isolated degree d vertices and leaves the number of edges constant. Since d is 2, 3, 4 or 5, the reduction can be implemented in time linear in the number of vertices removed.

(ii) Handling nonregular graphs with no isolated vertices of low degree

In this section we give the reductions for a graph with a vertex of degree d, d = 2, 3, 4, 5, which has at least one neighbor of degree d and at least one neighbor of degree x, x $\neq$ d. We leave one exception for the next section. That is, the case where d = 4 and the neighbors are of degree 4, x, 4, y: the two degree four neighbors being separated at the degree x and y neighbors.

We assign priorities to the reductions so as to remove all the degree 2 vertices first, then degrees 3, 4 and finally 5. Let v be a vertex of degree d having at least one neighbor of degree d and at least one neighbor of degree x $\neq$ d and assume v does not fit the exception mentioned above. Let C be the d-circle whose components are the degrees of the vertices in counterclockwise order about v with the change that every component not equal to d is set to x.[†] Then C can only be one of a finite number of d-circles. Furthermore, the only (circle) isomorphism for each of these d-circles onto itself is the identity isomorphism. This allows us to distinguish the edges at v. Specifically for each such degree d vertex we define an incident edge which is also incident to a vertex of degree other than d to be distinguished in such a manner that an isomorphism of G_1 onto G_2 mapping v onto w must map the distinguished edge at v onto the distinguished edge at w. This can be done because we need only define distinguished edges for a finite number of cases corresponding to classes of such degree d vertices, the class for a vertex v being determined by its circle C as defined above.

We summarize:

1. Vertices of low degree d having at least one neighboring vertex of degree d and at least one neighboring vertex of degree not equal to d can be classified in a finite number of classes.

2. The determination of the class of such a vertex requires only local information near the vertex.

3. Distinguished edges can be defined for each such class, and therefore each such vertex.

The following is the actual reduction for the degree d case:

Let v be a vertex of degree d and let (v,w) be a distinguished edge. Let (u,v) be counterclockwise adjacent to (v,w) in the ordering of edges about v. Construct the vector (w-label(w,v), v-label(w,v), label(v), v-label(u,v)) for each vertex v. Partition all vectors so constructed and assign integers. Replace the v-label of (u,v) by the

† x is handled as a special integer in the obvious manner.

appropriate integer. Let (w,w_1) be clockwise adjacent to (v,w) at w and let (w,w_k) be counterclockwise adjacent to (v,w) at w. Delete the edge (v,w) and the vertex v from the graph. Insert the edges formerly incident at v, in order starting with (u,v) into the adjacency list for w between (w,w_1) and (w,w_k).

There is a separate reduction as above for each of d = 2, 3, 4 and 5.

It is important to observe that no edge end which is moved or relabelled due to the reduction at v is affected by the same reduction at any other vertex. (Observe that the vertex w may be incident with several distinguished edges and that u may be degree d and have a distinguished edge of its own adjacent to (v,u) at u.) Furthermore, G_1 and G_2 are isomorphic, if and only if their reductions are identical.

The final observations about this reduction, that each d is fixed (2, 3, 4 or 5) and that an edge is collapsed and a vertex removed for each such vertex processed, allow us to conclude that this reduction runs in time linear in the decrease in edges and vertices.

(iii) <u>The first exception</u>

The special reduction for degree four vertices which are connected to two degree four vertices and two arbitrary non-degree four vertices in the counterclockwise circle ordering (4, x, 4, y) follows: For each vertex of this type v, let the neighboring vertices corresponding to (4, x, 4, y) be (r, s, t, u). Disconnect edges (r,v) and (t,v) at v and reconnect to vertices s and u, respectively. Remove vertex v and labels at v making (s,v), (v,u) into one edge (s,u). Associate with the s-label of (r,s) (old v-label of (r,v)) the vector (label (v), v-label(s,v), v-label(r,v)) and with the u-label of (t,u) (old v-label of (t,v)) the vector (label(v), v-label (u,v), v-label(t,v)).

After partitioning these vectors, relabel the labels with the number assigned to their associated vectors. Note that this reduction removes the equivalent of one vertex and one edge for every degree 4 vertex of this special type. Again we can conclude that this reduction runs in time linear in the decrease in edges and vertices.

The reader should observe that this is another reduction where an edge may be affected by the application of the same reduction at two different points in the graph. We leave it as an exercise to the reader to show that the resulting graph is the same regardless of the order of the points at which the reduction is applied.

(iv) <u>Handling regular graphs with vertices with solitary faces</u>

Given any planar graph, at least one of the reductions discussed so far is applicable except in the case of regular degree 3, 4 or 5 graphs. (Regular degree two graphs are n-gons which are handled by applying the circle isomorphism directly.)† Euler's theorem now guarantees the existence of a 3, 4 or 5 sided face. In fact, for regular graphs of degree 4 or 5 there must exist a three sided face. (We will give the details of a similar assertion later, to illustrate such proofs.) "Label" the faces 3, 4, 5 or x depending on whether they have 3, 4, 5 or greater than five edges.

Denote face F as <u>solitary</u> with respect to v if it is the only face with its label at a low degree vertex v. Let e and f be the edges of F incident at v with $eCCW_v f$. Then edge e is called the <u>distinguished</u> edge of face F at vertex v. These distinguished edges have unique solitary faces F and unique associated vertices v. In addition,

† Note that two circles (one clockwise, one counterclockwise) have to be formed for each n-gon in an analogous manner to skeins (its dual case).

determining whether a vertex v is a solitary face vertex and finding the distinguished edge requires only checking finite local information near v.

This next reduction is applied to regular graphs with solitary faces labelled d. (There are reductions for d = 3, 4, 5 and x.) For each vertex v having a distinguished edge with a solitary d-labelled face, the reduction collapses the distinguished edge and removes v as in (ii). However, there is added complication. Difficulty occurs because there may be several distinguished edges incident at a vertex. The reasons for this are that each edge of a solitary face labelled d may be a distinguished edge for some vertex, and that several faces of the same size may meet at a vertex v and be solitary faces for adjacent vertices. In order to visualize the most complicated situation, let F be a solitary face with label d with respect to v and let e be the distinguished edge. Treat the edge as a directed edge out of v. Then, the directed graph consisting of all distinguished edges satisfies the following two properties: There is at most one edge directed out of any vertex and if there is an edge directed out of a vertex there can be at most one edge directed in. Thus, the set of all distinguished edges forms a collection of trees and cycles. In the case of a cycle it is easily shown that the cycle is the perimeter of a solitary face.

We outline the reductions: The forest of trees is reduced in stages. The leaves of still existing trees are collapsed at each stage until all trees disappear to just points.[†] Then all remaining distinguished edges must be in cycles. These cycles are handled last, shrinking their solitary faces into points which are uniformly relabelled by a new integer. (Note that except for this integer, relabelling of remaining edges is the same in the previous general class of reductions.)

Once again an edge and vertex is removed for each vertex (and distinguished edge) processed. Again, this time, if we allow ourselves to think of the edges about vertices as circular lists which can be spliced together, the time cost for each vertex is fixed and again the reduction runs in time linear in the decrease in edges and vertices.

(v) The last general case

We have now only to deal with regular low degree graphs where some vertex v has incident faces with precisely two label values as defined previously, each label value occurring at least twice. (Otherwise we would have one of the regular polyhedral terminal cases or a graph with a solitary face.) This implies we can immediately restrict attention to regular graphs of degrees 4 or 5. For these graphs there must be faces with 3 edges (Euler). "Label" faces with greater than 3 edges with an x, and triangular faces with a 3. Consider the vertices v having three sided faces. Except for the special degree 4 case with circle (3, x, 3, x), the circles formed by the face labels of these v's consist of five distinct isomorphism classes (four of them degree 5, one degree 4). In each of these classes, there is either a unique pair of adjacent 3's or a unique pair of adjacent x's (or both). Define the edge separating the two faces corresponding to a unique pair to be distinguished in each of these classes. We conclude that distinguished edges can be defined for each of these vertices using finite local information near the vertex.

We outline the reduction for the case where the distinguished edge corresponds to a pair of adjacent 3-faces. Again the distinguished edges are collapsed as in (ii), but we may have the added possibility of this being a distinguished edge for both vertices

† At each of these stages, vectors are created and a partitioning assignment occurs.

of the edge in question. If this occurs we simply remove the edge, relabelling remaining edges as previously for (ii), collapse the two vertices into one and relabel this vertex to preserve the possible symmetry. (This is similar to the solitary face cycle case discussed earlier.) A similar reduction is defined for distinguished edges separating faces corresponding to adjacent x faces.†

A similar argument to previous cases allows us again to conclude that the reduction runs in time linear in the decrease in edges and vertices.

(vi) Another special degree 4 case

This special case corresponds to the face label circle of (3, x, 3, x) of the previous section. We develop this case to illustrate a particular reduction:

Note, first, that if the graph is regular degree four with opposite faces at each vertex identical in edge size, there can be faces of only two sizes and that Euler's theorem guarantees the smaller face size must be three as follows:

Let V_i represent the number of vertices with i edges incident and F_i represent the number of faces with i edges and E the number of edges.

In general V_i, $F_i \geq 0$ and F_1 and $F_2 = 0$ because of the removal of such cases by previous higher priority reductions.

We have $4V_4 = \sum_{i=3}^{\infty} iF_i = 2E$ or $V = V_4 = \frac{1}{4} \sum_{i=3}^{\infty} iF_i$. $V + F = E + 2$ yields:

$$\frac{1}{4} \sum_{i=3}^{\infty} iF_i + \sum_{i=3}^{\infty} F_i = \frac{1}{2} \sum_{i=3}^{\infty} iF_i + 2$$

or

$$F_3 - (F_5 + 2F_6 + 3F_7 + 4F_8 + \dots) = 8$$

so $F_3 > 0$.

This allows us to do the following: for each 3-face we can place a new vertex in the 3-face, connect the vertex to each of the 3-face vertices, remove the 3 edges of the 3-face labelling vertex and edges with encodings of a finite amount of local information sufficient to recover the 3-face.††

We can associate all the original vertices with the 3-faces by assigning $\frac{3}{2}$ of a vertex to each 3-face. It can be observed that one more vertex was added for each 3-face and the number of edges remained constant. If we examine the new resulting graph we can see that all original vertices which were degree 4 are now degree 2 and that each newly added vertex is degree 3. The degree 2 vertices can all be removed,* reducing the number of vertices by $\frac{3}{2}$ of a vertex for each original 3-face, and the edges by a half. We can now tally all the vertices and edges and see that we have $\frac{2}{3}$ as many vertices and half as many edges as originally present. The above reduction can be implemented in time linear in the number of vertices of the original graph as it was before this reduction. So once again we can conclude that the reduction just specified runs in time linear in the number of vertices and edges removed.

We complete this section on the list of reductions with the final observation that though the list is long, the total number of reductions is finite and fixed. The next section provides a rough outline of the data structures for an implementation of the algorithm.

† This yields a total of 3 reductions (two for degree 5, one for degree 4).

†† The reader may verify that this can be done.

* Apply isolated degree 2 removal, followed by clump reduction.

4. Data Structures

Two main items must be described by the data structures: one is the current graph representation; the other, the current list of applicable reductions. The data structures must have the property of being easily modified to reflect changes caused by the application of a reduction; that is, transformations of the graph representations, and updates of the reductions list.

Other data structures which will not be detailed as precisely act as temporary storage for data (or pointers to data) during the reduction and updating process.

A. Specifying the graph

To specify the graph, each vertex and each face is given a number which acts uniquely as its "name." Edges are specified by their two edge-ends. That is, each edge-end is given a number. The edge-end at a vertex is said to be incident with the vertex. Three scalar variables #FACE, #VERTICES, #EDGE_ENDS give respectively the current number of faces, vertices, and edge-ends. Associated with vertices are arrays VLABEL, DEGREE, VEDGE which when indexed by a vertex name yield respectively the label, degree and one edge-end incident at that vertex. Similarly for faces, EDGE# and FEDGE yield the number of EDGES about a face, and one counterclockwise edge-end of an edge from the face's perimeter.

Edge-ends have the arrays ELABEL, VERTEX, CCWFACE, CCWEDGE, CWEDGE, and OTHEREDGE, specifying the label of the edge-end, the name of the vertex at which the edge-end is incident, the name of the face between it and its counterclockwise adjacent edge-end, the name of the counterclockwise adjacent edge-end, the name of the clockwise adjacent edge-end, and the other edge-end of its edge, respectively. The clockwise face CWFACE can be obtained by consulting the CCWFACE entry of the other edge-end.†

B. Specifying the current list of reductions

Each reduction is given a number in the order of its priority. The array REDUCTION acts as an array of headers, each header pointing to a list of items ITEM to which that particular reduction can be applied. These lists are doubly-linked so that an item can be removed from a list without changing the memory locations of any other item. Besides these links each item also has fields IVERTEX and IEDGE_END, which give the names of the vertex and edge-end they are most closely associated with. This is fixed for a reduction, but varies as to the type of reduction. To illustrate: for loops the vertex is the loop vertex and the edge is the non-loop edge; for degree one vertices the vertex is the spoke center and the edge is the non-spoke edge; and for vertices of degree 2, 3, 4 and 5 the vertex is itself and the edge is the distinguished edge. Another field in the item gives the reduction name. Note that these vertices and edges appear at most once in the reduction list. When they appear, they do so under the highest priority reduction for which they qualify.

There is an array VITEM which when indexed by the name of a vertex gives the location of the ITEM (if it exists) which has this vertex in its IVERTEX field. If this does not exist, it contains a specified value meaning undefined. There is a similar array EITEM for edge-ends.

† These structures can be constructed in linear time from the adjacency lists for the graphs. For assistance, see references in [2,3].

C. General outline

The general outline for the reduction algorithm is now clear: prior to each reduction application, the REDUCTION array is scanned for the first (highest priority) non-empty list of items. (This can be done easily by giving an array which tells you the current number of items in each list.) This reduction is then "applied," that is, the graph representing data structures are modified to reflect application of this reduction. At the same time pointers to vertices, edge-ends, faces which are modified or whose local conditions have changed, are stored. On a subsequent pass over these pointers, items in the reduction lists are removed and added to reflect the new relationships. This updating can be done in time linear in the decrease in complexity of the graph, the decrease affected by the prior reduction.

D. More details on changing graph representation

(i) vertex and edge labels

Most new vertex and edge labels come from encodings of vectors. Along with each such vector there is associated a pointer back to the vertex or edge-end. Applying the vector partitioning algorithm results in lists of identical vectors to which the new "names" can be assigned. Their pointers in turn specify immediately which labels are to be updated.

(ii) edge information for faces and vertices

EDGE# should be updated with the removal of each edge for the faces involved. Similarly DEGREE for the vertices involved.

checking FEDGE and VEDGE

Before an edge is removed, FEDGE and VEDGE of the associated two faces and two vertices should be checked and changed if necessary to point to another, not yet removed edge of these faces (in the case of faces) and vertices (in the case of vertices). (If there is no such edge, the edge-associated face and edge-associated vertex can be removed completely.)

(iii) edge-end modifications

Entries for an edge in CWFACE and CCWFACE remain valid as long as the edge is still around. The entries in CCWEDGE and CWEDGE for edge-ends incident at each vertex form doubly-linked circular lists around the vertex. When the collapsing of an edge occurs, two such lists are cut and spliced to form one large one.

The only modification which could cause problems is changing the vertex pointer VERTEX. However, except for one case, all the edge collapsing that occurs in this case involves vertices of degree ≤ 5 into vertices with degree possibly > 5. This one case is the solitary face vertex reduction. There, if we are careful to update VERTEX for each edge-end transfered only once at the end of the reduction modifications, rather than in stages, we can account for the cost of the VERTEX updates in our linear time bound.

5. Concluding Remarks

An extension of this algorithm can, by storing the reduction history of the graphs, give us a precise isomorphism mapping if it exists. Also as the reader has perhaps already noticed, there is also an obvious dual (face-oriented) version.

In conclusion, we think this planar graph isomorphism algorithm's importance is mostly theoretical, demonstrating existence rather than providing a practical algorithm; its relative inelegance seems to suggest that "better" perhaps even practical linear algorithms exist and that the problem is still not yet fully understood. The ideas of the algorithm might be extendible to reducing the states of and determining isomorphism for special cases of finite automata.

References

[1] Cook, S.A., "Linear time simulation of deterministic two-way pushdown automata," IFIP Congress 71: Foundations of Information Processing, Proceedings, Ljubljana, Yugoslavia, August, 1971; 174-179, North-Holland Publishing Company, Amsterdam, 1972.

[2] Hopcroft, J.E., and R.E. Tarjan, "A V log V algorithm for isomorphism of triconnected planar graphs," JCSS 7:3 (1973), 323-331.

[3] Hopcroft, J.E., and R.E. Tarjan, "Isomorphism of planar graphs (working paper)," in Complexity of Computer Computations, (R.E. Miller and J.W. Thatcher (eds.), 143-150, Plenum Press, New York, 1972.

[4] Ore, O. The Four Color Problem, Academic Press, New York, 1967, 48-61.

[5] Weinberg, L., "A simple and efficient algorithm for determining isomorphism of planar triply connected graphs," IEEE Trans. on Circuit Theory 13(1966), 142-148.

[6] Whitney, H., "A set of topological invariants for graphs," Amer. J. Math. 55(1933), 321-235.