

# Short Encodings of Planar Graphs and Maps

Kenneth Keeler\*  
Jeffery Westbrook†

August 3, 1993

## Abstract

We discuss space-efficient encoding schemes for planar graphs and maps. Our results improve on the constants of previous schemes and can be achieved with simple encoding algorithms. They are near-optimal in number of bits per edge.

## 1 Introduction

In this paper we discuss space-efficient binary encoding schemes for several classes of unlabeled connected planar graphs and maps. In encoding a graph we must encode the incidences among vertexes and edges. By maps we understand topological equivalence classes of planar embeddings of planar graphs. In encoding a map we are required to encode the topology of the embedding *i.e.*, incidences among faces, edges, and vertexes, as well as the graph. Each map is an embedding of a unique graph, but a given graph may have multiple embeddings. Hence maps must require more bits to encode than graphs in some average sense.

There are a number of recent results on space-efficient encoding. A standard adjacency list encoding of an unlabeled graph  $G$  requires  $\Theta(m \lg n)$  bits, where  $m$  and  $n$  are the number of edges and vertexes, respectively. Turán [9] gives an encoding of unlabeled connected planar graphs and maps which uses (asymptotically)  $4m$  bits<sup>1</sup>. Itai and Rodeh [5] give a scheme for labeled planar graphs requiring  $\frac{3}{2}n \lg n + O(n)$  bits, and Naor [7] gives a method for general unlabeled graphs which uses  $n^2/2 - n \lg n + O(n)$  bits (the storage requirement is shown to be optimal to second order). We may also mention that Jacobson [6] gives an  $\Theta(n)$  space

---

\*Parts of this research were performed while this author was with the Division of Applied Sciences, Harvard University (supported by U.S. Army Research Office Contract DAAL03-86-K-0171) and the Performance Analysis Department, AT&T Bell Laboratories.

†Department of Computer Science, Yale University, New Haven, CT 06520-2158. Research partially supported by National Science Foundation Grant CCR-9009753.

<sup>1</sup>This is not actually stated in [9]; the storage requirement is given as  $\leq 12$  bits per vertex.

encoding of unlabeled connected planar graphs which supports traversal in  $\Theta(\lg n)$  time per vertex visited. The constant factor in the space bound is relatively large, however.

Our encoding schemes for planar graphs are at heart schemes for encoding maps: we choose a particular planar embedding and encode the resulting map. This is the procedure as well in references [6, 9].

Following Tutte [11], we allow graphs and maps to have multiple edges between two vertexes, and to contain *loop* edges, which are edges whose endpoints coincide. Graphs and maps may also contain degree-one vertexes; we call such a degree-one vertex and its incident edge a *stick*. The presence of loops and sticks affects how compactly we can encode the graph. The natural measure of map size is the number of edges, and this quantity governs the size of our encodings even though planar graph size is typically measured by number of vertexes. All maps and graphs to be encoded in the rest of this paper shall be understood to be unlabeled and connected.

We show encodings for

- loop- and stick-free maps and graphs in 3 bits per edge. Two important subclasses of this class are the 2-connected maps and graphs.
- arbitrary planar graphs in  $\lg 12$  bits per edge,
- arbitrary maps without loop edges in  $\lg 12$  bits per edge,
- arbitrary maps without stick edges in  $\lg 12$  bits per edge,
- proper planar triangulations in  $(3 + \lg 3)/2$  bits/region or  $(3 + \lg 3)/3$  bits per edge.

The unusual constant  $\lg 12 \approx 3.58$  is of particular significance in view of Tutte’s enumeration of the rooted connected planar maps with  $m$  edges [11]. A rooted map is a map in which one edge has been designated the root. There are

$$A_m = \frac{2(2m)!3^m}{m!(m+2)!}$$

rooted connected maps and between  $A_m/4m$  and  $A_m$  unrooted connected maps. By way of Stirling’s approximation we have

$$A_m = 12^m \cdot 2^{-\frac{5}{2}\lg m + O(1)}(1 + O(1/m)).$$

Then given any code for maps with  $m$  edges, the number of maps whose codewords are shorter than  $m(\lg 12 - \epsilon) + O(1)$  bits is  $o(A_m/m)$  for any  $\epsilon$ .<sup>2</sup> Another enumeration due to Tutte [10] gives the comparable bound for triangulations, with which we shall compare our

---

<sup>2</sup>This is a sharper version of what is often called in the computer science literature the “information-theoretic lower bound,” by which is meant simply the logarithm of the size of the set to be encoded.

result. In this paper we deal with unrooted maps, but the encoding schemes work equally well with rooted maps.

Our goal in this paper is primarily theoretical: to move towards encoding schemes for planar maps that are the shortest possible according to Tutte's results. Besides showing the existence of compact encoding schemes, however, it is also important to give efficient encoding and decoding algorithms to convert between a standard adjacency list representation and the space-efficient encoding. Our encoding and decoding algorithms are simple, run in linear time, and provide the most compact encoding currently known, so our results may have some practical application. In addition, our schemes immediately imply an encoding for labeled planar graphs that requires  $n \lg n + m \lg 12 + o(n)$  bits, thus improving on Itai and Rodeh.

Our basic idea is to construct a particular depth-first search tree of a map, then sequentially to delete the non-tree edges and add one-bit labels to the tree edges in such a way that the non-tree edges can be reconstructed from the labels. This converts the map into a labeled version of the search tree. We then encode the tree in any standard way, followed by an encoding of the string of labels.

Harary [3] is a reference for the graph theoretic terminology we use in this paper.

## 2 Encoding Loop- and stick-free Maps in 3 Bits/Edge

In this section we describe an encoding scheme for maps that contain neither loop edges nor sticks. Since every map embeds a unique planar graph, this scheme also suffices to encode loop- and stick-free planar graphs.

A *topological* depth-first search (TDFS) of a connected graph  $G$  embedded in an oriented surface (or, to abuse definitions slightly, of a map  $M$  in that surface) is a depth-first search [8] in which vertexes adjacent to the current vertex are recursively searched in counter-clockwise (CCW) order of the corresponding edges around the current vertex, starting at the edge from the current vertex to its parent. (In standard DFS, the adjacent vertexes are searched in arbitrary order.) The TDFS is started at a *root edge*  $\langle u, v \rangle$ , choosing one of  $u$  or  $v$  to be the root vertex of the TDFS tree and searching first along  $\langle u, v \rangle$ . The root edge and root vertex are chosen arbitrarily unless otherwise specified. By convention the exterior face of the map is taken to be that face lying between the root edge and its predecessor in CCW order around the tree root.

If  $G$  has  $n$  vertexes and  $m$  edges then  $T$  has the same  $n$  vertexes and  $n - 1$  of the  $m$  edges. There are therefore  $k = m - n + 1$  *non-tree edges*, the set of which we denote by  $N$ . Depth-first search has the property that for any non-tree edge  $\langle u, v \rangle \in N$ , either  $u$  is an ancestor of  $v$  or *vice versa* [8]. Figure 1 gives an example of a map  $M$  and Figure 2 shows the TDFS  $T$  of  $M$  when the root node is 1 and the root edge is  $\langle 1, 2 \rangle$ .

**Lemma 2.1** *Let  $e = \langle u, v \rangle$  be a non-tree edge, and assume that  $u$  is the ancestor of  $v$ . Let  $f$  be the last edge on the TDFS tree path from  $v$  to  $u$ . Let  $t$  denote the edge from  $u$  to its*

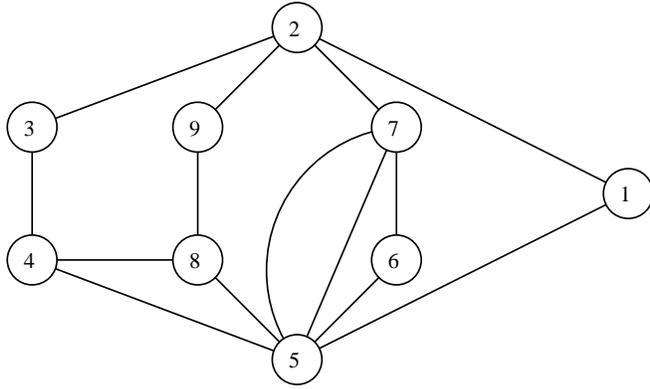


Figure 1: A map  $M$ .

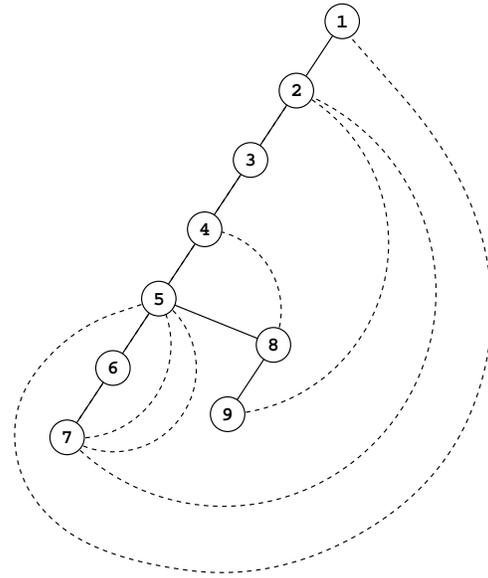


Figure 2: A TDFS of map  $M$ . Non-tree edges are shown dashed.

parent, or the root edge if  $u$  is the root vertex. Then  $f$  precedes  $e$  in counterclockwise order around  $u$  starting at edge  $t$ .

**Proof:** The edges out of  $u$  were explored in counterclockwise order starting at edge  $t$ . If  $e$  occurred before  $f$ , then  $v$  would have been made a child of  $u$  by edge  $e$ , contradicting the assumption that  $e$  is a non-tree edge. ♣

Let  $M$  be a loop-free and stick-free map with  $n$  vertices and  $m$  edges. Briefly to sketch our encoding method: we first compute a TDFS tree  $T$  of  $M$ . We perform simple modifications, converting  $M$  and  $T$  to a new map  $M_0$  and TDFS tree  $T_0$ . We then label each edge of  $M_0$  with a “0” and compute a series of maps  $M_i$ ,  $i = 1, \dots, k$ , at each stage deleting an edge and changing some labels. Each map  $M_i$  will have the same TDFS tree  $T_0$ .  $M_k$  will be a copy of  $T_0$  whose edges have been labeled with either a “0” or “1”. Finally we encode  $T_0$  by encoding the tree in a standard way and append the associated labels as a string.

To convert  $M$  and  $T$  to  $M_0$  and  $T_0$ , we examine each non-tree edge  $e = \langle u, v \rangle \in N$ . Assume  $u$  is the ancestor of  $v$  in  $T$ . We split  $e$  by inserting a degree-2 node  $w_e$ , thereby creating two edges  $\langle u, w_e \rangle$  and  $\langle w_e, v \rangle$ . We make  $w_e$  a child of  $v$  by adding edge  $\langle v, w_e \rangle$  to  $T_0$ . This gives a new embedded graph  $M_0$  with TDFS tree  $T_0$  and non-tree edges  $\{\langle w_e, v \rangle, e \in N\}$ . The new tree  $T_0$  has  $n + k = m + 1$  vertices and  $m$  edges. Since  $M$  contains no sticks, every leaf in  $T$ , the initial TDFS tree, has at least one incident non-tree edge. Hence the leaves of the modified tree  $T_0$  consist exactly of the  $k$  new nodes  $w_e, e \in N$ , the set of which we denote by  $W$ . Figure 3 shows the modified tree  $T_0$  for the graph of Figure 1. New nodes  $w_e$  are shown as rounded squares.

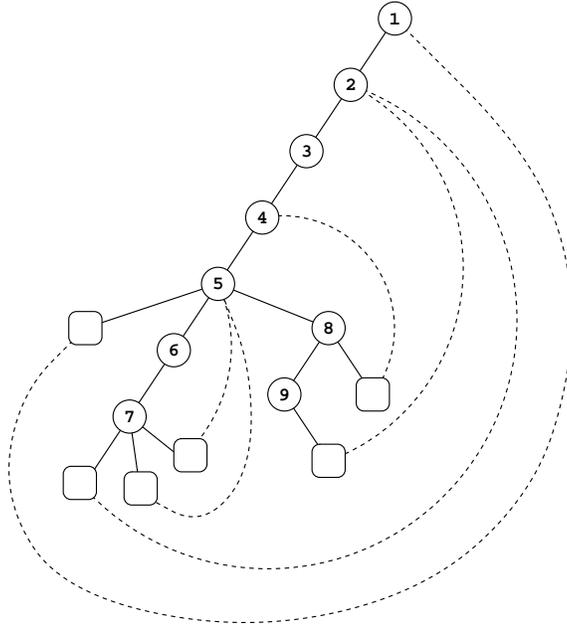


Figure 3: TDFS tree  $T_0$  of  $M_0$ .

Let  $l_1, l_2, \dots, l_k$  be the leaf nodes of  $T_0$  in order from left to right, i.e., by increasing pre-order number. We compute  $M_i$  from  $M_{i-1}$  by processing leaf  $l_i$ . Let leaf  $l_i$  have parent  $v$  and incident non-tree edge  $e = \langle l_i, u \rangle$ . Let  $f = \langle l_i, v \rangle$  and let  $g$  be the edge that immediately precedes  $e$  in CCW order around  $u$ . (As shown below, following Lemma 2.2, we are guaranteed that  $f \neq g$ .) Using these definitions we process leaf  $l_i$  as follows.

1. Label edge  $g$  with a “1”.
2. Label edge  $f$  with the current label of edge  $e$ .
3. Delete edge  $e$  from  $M_{i-1}$  to form  $M_i$

Figure 4 shows the tree  $T_1$  for the graph of Figure 1, after processing leaf 1 and prior to processing the second leaf. Edges  $f$  and  $g$  are indicated.

We now establish a series of lemmas that lead us to the encoding scheme.

**Lemma 2.2**  $M_k$  is a tree.

**Proof:** By construction, every non-tree edge in  $M_0$  is of the form  $\langle w_e, u \rangle$  for  $w_e \in W$  and  $u \in V$ .  $M_k$  is constructed by examining each leaf in turn and deleting the incident non-tree edge. ♣

Consider processing leaf  $l_i$  with incident edge  $e \in N$ . Define the tree path from leaf  $l_i$  to  $u$  to be  $P(i) = (l_i, v = \nu_1, \nu_2, \dots, \nu_\kappa = u)$ . We have  $\kappa \geq 2$ , since otherwise  $u = v$ , implying  $l_i$

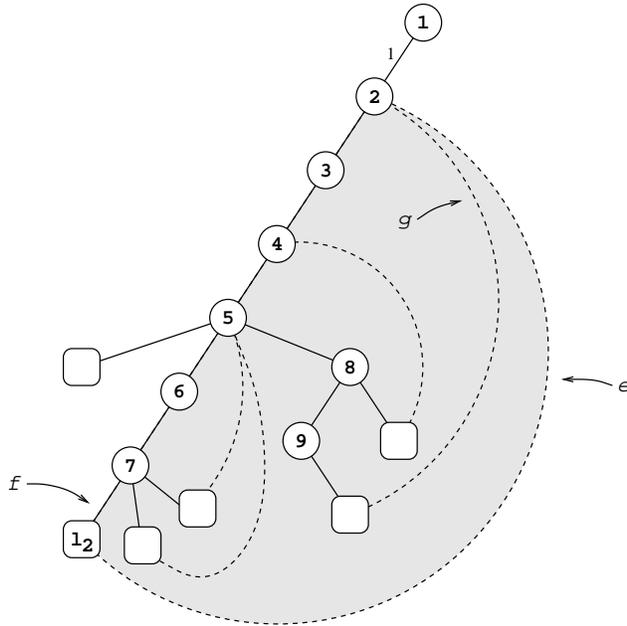


Figure 4: Tree  $T_1$  prior to processing leaf  $l_2$ . (Edges have label “0” unless marked “1”.) Shaded area is  $INT(2)$ .

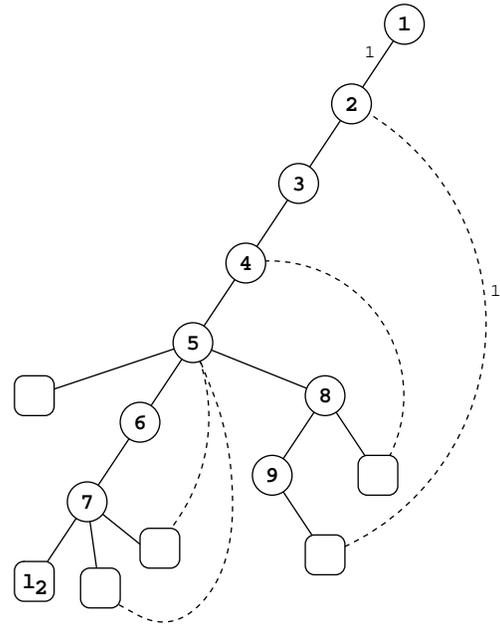


Figure 5: Tree  $T_2$  after processing leaf  $l_2$ .

must have been inserted into a loop edge, contradicting the assumption that  $M$  is loop-free.

The cycle formed by  $e$  and  $P(i)$  separates the plane into an interior and an exterior region, dividing the vertexes and edges of  $M_{i-1} - (P(i) \cup \{e\})$  into an interior set  $INT(i)$  and an exterior set  $EXT(i)$ . By planarity, no edge of  $M_{i-1}$  connects a vertex of  $INT(i)$  with one of  $EXT(i)$ .

**Lemma 2.3** Consider leaf  $l_i$ . Let  $l_j$  be a leaf node in  $INT(i)$ . Then  $j > i$ .

**Proof:** By planarity, the tree path from  $l_j$  to the root must intersect the boundary of  $INT(i)$  at some node  $\nu_\gamma \in P(i)$ . Thus  $\nu_\gamma$  is the least common ancestor of  $l_i$  and  $l_j$ . Furthermore, since the first edge on the path from  $\nu_\gamma$  to  $l_j$  must lie inside  $INT(i)$ , it must follow the edge  $\langle \nu_{\gamma-1}, \nu_\gamma \rangle$  in CCW order around  $\nu_\gamma$ . But then a TDFS will visit and number  $l_i$  before it numbers  $l_j$ . ♣

**Lemma 2.4** In  $M_{i-1}$ , all edges connecting vertexes in  $INT(i) \cup P(i)$  are labeled “0”.

**Proof:** Suppose that there is an edge  $f$  contained in  $INT(i)$  or on  $P(i)$  that has a label other than “0”. Edge  $f$  received this non-zero label during the previous processing of some leaf  $l_j$ ,  $j < i$ . It received this label either because it is incident on  $l_j$  (Step 2) or because it immediately precedes the non-tree edge  $f' = \langle l_j, u \rangle$  in CCW order around  $u$  (in Step 1).

In either case this requires that  $l_j$  lie in  $INT(i)$  or  $P(i)$ : in the former case this follows by hypothesis, and in the latter case planarity implies that edge  $f'$ , which is incident on  $u$  and immediately follows  $f$  in CCW order, must lie entirely in  $INT(i)$ . But, since  $j < i$ ,  $l_j$  cannot lie in  $INT(i)$  because of Lemma 2.3, and  $l_j$  cannot belong to the tree path  $P(i)$ , since  $l_i$  is the only leaf on  $P(i)$ . ♣

**Lemma 2.5** *Each edge of  $M$  is given a non-zero label at most once.*

**Proof:** Consider the processing of leaf node  $l_i$ . The two distinct edges  $f$  and  $g$  labeled in this processing are both in  $INT(i) \cup P(i)$ , so Lemma 2.4 implies that both are previously labeled “0”. ♣

**Lemma 2.6** *Map  $M_{i-1}$  can be constructed from map  $M_i$ .*

**Proof:** We simply reverse the processing of leaf  $l_i$  that created  $M_i$  from  $M_{i-1}$ . The non-tree edge  $e$  deleted in Step 3 of this processing satisfied the following:

1.  $e$  connected  $l_i$  to some ancestor  $u$  other than the parent  $v$  of  $l_i$ ;
2.  $e$  immediately followed an edge  $g$  marked “1” such that  $g$  follows in CCW order the last tree edge on the path from  $l_i$  to  $u$  (or  $g$  is itself that tree edge);
3.  $e$  enclosed a region in which all edges were marked “0” except for  $g$  and possibly the edge  $f$  from  $l_i$  to its parent (see Lemma 2.4).

These conditions uniquely determine where edge  $e$  must be inserted. For suppose there were two choices  $g_1$  and  $g_2$  satisfying the conditions. By condition 1  $g_1$  is incident on some vertex  $u_1$  and  $g_2$  is incident on  $u_2$  such that  $u_1$  and  $u_2$  are ancestors of  $l_i$ . If  $u_1 = u_2$ , then assume without loss of generality that  $g_2$  follows  $g_1$  in CCW order. But then if we attach  $e$  as a successor to  $g_2$ , the resulting region  $INT(l_i)$  contains  $g_1$ , violating condition 3. Hence  $g_2$  is not a valid choice. On the other hand, suppose without loss of generality that  $u_2$  is an ancestor of  $u_1$ . Again, if we attach  $e$  as a successor to  $g_2$ , the resulting region  $INT(l_i)$  contains  $g_1$ .

After the non-tree edge  $e$  is inserted, we undo Steps 2 and 1 by copying the label of  $f$  to  $e$  and labeling  $f$  and  $g$  with “0”. ♣

**Theorem 2.7** *A 2-connected planar graph  $G$  with  $m$  edges can be encoded in  $3m + O(1)$  bits.*

**Proof:** We take  $M$  and apply the series of transformations described above to produce the graph  $M_k$ , which is a tree of  $m + 1$  nodes whose edges are labeled “0” or “1”. To encode this we traverse the edges of  $M_k$  in depth-first order and write down the labels in a string  $\sigma_2$ .

Simultaneously, we encode the unlabeled tree  $M_k$  in a string  $\sigma_1$  by writing a “1” whenever the traversal descends an edge and a “0” when it subsequently ascends that edge. This well-known representation requires 2 bits per edge. We concatenate  $\sigma_1$  and  $\sigma_2$  to give the encoding  $\sigma_1 \circ \sigma_2$  of map  $M$ .

That this is uniquely decodable is clear: given  $m$ , we can read off  $\sigma_1$  and  $\sigma_2$  and build the labeled tree  $M_k$ ;  $M_0$  can then be built by applying Lemma 2.6 repeatedly, and  $M$  is then recovered by replacing all nodes which were leaves in  $M_k$  by single edges.

If  $m$  is not taken as known by the decoder, a degree-one node can be added to  $T$  whose sole child is the original root of  $T$ . The string  $\sigma_1$  that encodes this modified tree can be extracted from the front of  $\sigma_1 \circ \sigma_2$  by observing that the encoding traversal must start and end at the degree-one root. Alternately,  $m$  may be prepended to  $\sigma_1 \circ \sigma_2$  with  $\sim \lg m$  bits using, *e.g.*, Elias’s representation of the positive integers [2]. This adds a term  $O(\log m)$  to the length of the encoding. ♣

Next we consider the efficiency of the encoding and decoding algorithms. We assume we begin and end with an adjacency list representation of a map, in which the edge lists for each vertex are ordered in CCW order. Each element in an edge list contains a pointer to an edge; each edge contains two backpointers to the corresponding elements in the adjacency lists of its two endpoints. This standard representation can be constructed from other standard representations such as the winged-edge data structure in  $O(m)$  time. Note that a planar graph can be embedded in linear time [1, 4].

The encoding algorithm is straightforward. The topological depth-first search runs in linear time, since the adjacency lists are ordered in CCW order. The subsequent tree traversal is also easily implemented in linear time, again using the CCW ordering of adjacency lists.

Decoding in linear time is only slightly more complex. We begin by reconstituting the labeled tree. We traverse the tree in reverse depth-first search manner, thus encountering the leaves in right-to-left rather than left-to-right order. We push tree edges onto a *main* stack as they are encountered going downward, and pop them as they are encountered going upward. We also maintain a *substack* of edges labeled “1”. When a tree edge labeled “1” is descended, it is pushed onto the substack. When a leaf node is encountered, the substack is popped. The non-tree edge corresponding to the leaf is inserted following the popped edge in CCW order around the popped edge’s parent endpoint. If the inserted non-tree edge is labeled “1,” it is then pushed onto the substack.

We argue by induction on the number of steps that first, the substack contains exactly the set of edges labeled “1” that are incident on ancestors of the current node in the DFS; and second, top to bottom order on the stack corresponds to least to greatest ancestor order in the tree. Note that we define a node to be its own least ancestor. If this holds, the insertion of each new non-tree edge is correct according to the proof of Lemma 2.6.

Suppose that the properties hold after  $k$  steps. Hence the decoding is correct up to the current step. An easily verified property is that at any intermediate step in a correct decoding, each edge labeled “1” is incident on an ancestor of some unprocessed leaf.

If we descend an edge and push it onto the substack, the properties remain true. Similarly, popping an edge and pushing a replacement during the processing of a leaf preserves the properties. Finally, we argue that in ascending some tree edge  $e = \langle u, v \rangle$ , such that  $v$  is the parent of  $u$ , it cannot be the case that the top edge on the substack is not incident on an ancestor of  $v$ . Since the property held prior to ascending, the hypothetical labeled edge must have been incident on  $u$ . But  $u$  is not an ancestor of an unprocessed leaf, violating the inductive hypothesis of correctness.

### 3 Planar Graphs in $\lg 12$ Bits per Edge

In this section we extend the technique of Section 2 to handle an arbitrary connected planar graph  $G$ . As before, the graph  $G$  is embedded, giving a map  $M$ , and a TDFS is performed on  $M$ , generating a tree  $T$ . This tree can have two structures that could not arise in the case of a 2-connected graph. First,  $T$  may have leaves without incident non-tree edges. Such a leaf corresponds to a degree-one vertex in  $G$ , and is called a *stick*. Thus in the general case we must distinguish between leaves corresponding to sticks and “regular” leaves  $w_e$  resulting from splitting non-tree edges.

Second,  $T$  may contain loop edges. In  $T_0$ , the tree produced from  $T$  by splitting non-tree edges, a loop edge  $e$  with endpoint  $v$  produces a leaf  $w_e$  with parent  $v$  and incident non-tree edge  $\langle w_e, v \rangle$ . This invalidates Lemma 2.5, since the edge  $g$  preceding  $e$  around its ancestor endpoint is no longer necessarily distinct from the tree edge  $f$  that connects  $w_e$  to its parent.

Given an embedding,  $M$ , of  $G$ , we produce a modified embedding,  $M'$ , in which loops and sticks are easy to handle. We begin by computing a TDFS tree,  $T$ , of  $M$ . Next, we rearrange the embedding so that each loop edge of  $M$  is “empty”, *i.e.*, its interior consists of a single face, and so that for each node  $v$  in  $T$ , the incident loop edges and sticks occur last in CCW order around  $v$ , starting at the edge from  $v$  to its parent in  $T$  (or the root edge if  $v$  is the tree root). The order of sticks and loops among themselves is arbitrary. Since sticks and loops are 1-connected components, this rearrangement can always be performed. Furthermore, the resulting tree  $T'$  is a valid TDFS of the new embedding  $M'$ . Figure 6 gives an example of a map  $M$  and Figure 7 shows a TDFS tree after rearranging sticks and loops. The critical consequence of the embedding rearrangement is that no non-tree edge follows a loop or stick in CCW order around the ancestor endpoint, and hence the loop or stick cannot receive a mark during the removal of some other non-tree edge.

Once the embedding has been rearranged, we proceed as in Section 2, splitting each non-tree edge  $e$  to give a new leaf vertex  $w_e$ , then processing each leaf  $l_i$  in order from left to right. If  $l_i$  is a leaf resulting from splitting a non-loop non-tree edge, then it is processed as in Section 2. Suppose  $l_i$  is the degree-one endpoint of a stick, or a degree-two leaf  $w_e$  resulting from splitting loop edge  $e$ . Let  $f$  be the tree edge from  $l_i$  to its parent. If  $f$  is a stick, it is labeled “\*0”. If  $f$  is half of a loop edge, the non-tree half of the loop edge is deleted, and  $f$  is labeled “\*1”. Thus the edge labels are now drawn from the ternary alphabet  $\{0, 1, *\}$ .

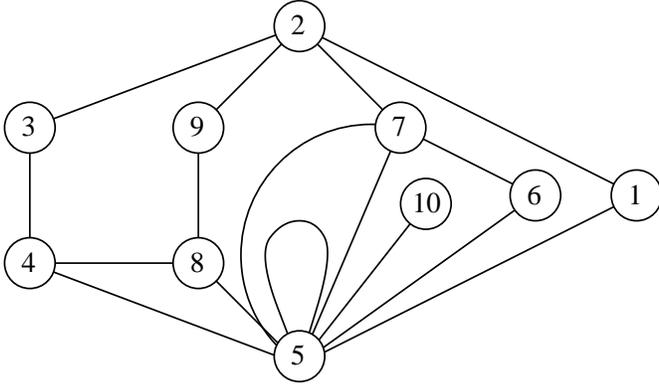


Figure 6: A map  $M$  with a stick (vertex 10) and a loop.

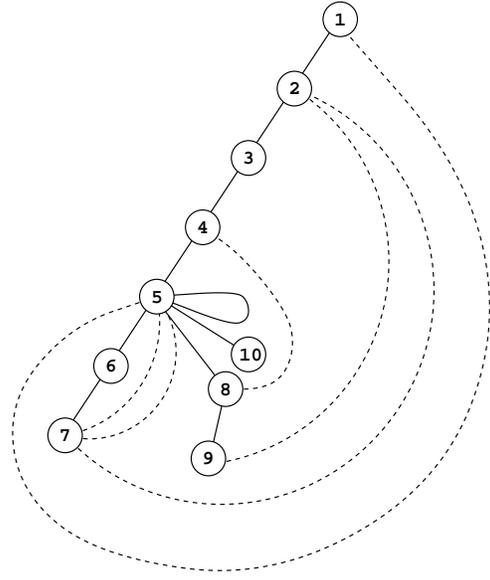


Figure 7: A TDFS of map  $M$  after rearrangement of the embedding.

**Theorem 3.1** *An arbitrary planar graph  $G$  with  $m$  edges can be encoded in  $m \lg 12 + O(1)$  bits.*

**Proof:** We encode the final labeled tree as in Section 2, traversing the tree in depth-first order, writing down the string of edge labels and encoding the structure of the tree by writing a “1” whenever an edge is descended and a “0” whenever it is subsequently ascended. If an edge leads to a leaf, the descending “1” is immediately followed by the ascending “0”. If an edge is labeled “\*0” or “\*1” then it must lead to a leaf, and the ascending “0” for that edge is redundant. We replace the ascending “0” with either a “0” or “1”, according to whether the edge label is “\*0” or “\*1”, respectively, and reduce the edge label to “\*”. In decoding, the tree string and the label string are read in lock-step: whenever a “1” is read off the tree string, indicating a new descending edge, the next label is read off the label string. If the label is “\*”, then the bit following the “1” in the tree string is appended to “\*” to give the correct edge label. This technique of storing the stick and loop labels in two places saves overall space.

Given  $k$ , it is possible to encode a string of length  $k$  over a *ternary* alphabet in a uniquely decodable way with  $k \lg 3 + O(1)$  bits. Regard the string as the standard ternary representation of an integer between 0 and  $3^k - 1$ ; the standard binary representation of this integer has length  $\lceil \lg 3^k \rceil = k \lg 3 + O(1)$  bits.

There are  $2m + O(1)$  bits to encode the tree plus  $m \lg 3 + O(1)$  bits to encode the label string, for a total of  $(2 + \lg 3)m + O(1) = m \lg 12 + O(1)$  bits. ♣

We are unable to achieve the upper bound of Theorem 3.1 in encoding arbitrary maps,

since in encoding maps we are not free to change the embedding. We can, however, match it in two important special cases.

**Corollary 3.2** *A loopless map  $M$  of  $m$  edges can be encoded in  $m \lg 12 + O(1)$  bits.*

**Proof:** Run the algorithm of Section 2 on map  $M$ , not processing leaves that result from sticks but remembering which they are. In the resulting labeled tree, stick edges are labeled “0” or “1.” Now examine each stick: replace the original “0” or “1” label by “\*0” or “\*1”, respectively. Encode the final tree, storing the second half of each stick label in place of the redundant ascending 0 as in the proof of Theorem 3.1. ♣

**Corollary 3.3** *A stick-free map  $M$  of  $m$  edges can be encoded in  $m \lg 12 + O(1)$  bits.*

**Proof:** Replace each empty loop edge by a stick. Now process  $M$  as in Corollary 3.2. In decoding, replace sticks by empty loops. Furthermore, modify the procedure described in Lemma 2.6 so that a non-tree edge can connect a leaf  $l_i$  to its parent. (This handles non-empty loop edges.) The correctness of this encoding scheme can be seen by examining Lemmas 2.2 through 2.6. ♣

To conclude the section, we remark that our techniques can be used to encode labeled planar graphs in  $n \lg n + m \lg 12 + O(1)$  bits by first encoding the structure of the graph using the appropriate encoding scheme described above and then writing down the string of labels in pre-order.

## 4 Triangulations

We can improve significantly upon Theorems 2.7 and 3.1 if the planar map  $M$  to be encoded is a proper triangulation, *i.e.*, each face of the embedding is a triangle consisting of three distinct edges. (The graph  $G$  underlying such a triangulation can be embedded in the plane in exactly one way up to the choice of exterior region, so  $M$  and  $G$  are in 1-1 correspondence.)

Let  $M^*$  be the planar dual of  $M$ . The graph  $G^*$  underlying  $M^*$  is regular of degree three and is two-connected. The map and its dual uniquely determine each other; the regularity of  $G^*$  implies that any TDFS tree  $T^*$  is a binary tree, which fact can be exploited for efficient encoding.

Let  $r$  be the number of regions in  $M$  and so the number of vertexes in  $M^*$  and the number of internal nodes in  $T^*$ . There are  $m = 3r/2$  edges in  $M$  and  $M^*$ , of which  $k = 3r/2 - (r - 1) = r/2 + 1$  are non-tree edges.

To proceed: choose an edge of  $M^*$  arbitrarily, and split it by inserting a new vertex  $s$ . Compute a (binary) TDFS tree  $T^*$  rooted at  $s$ . Each node  $v$  other than  $s$  then has an incident parent edge. The remaining two edges incident on  $v$  are called *first* or *second* according to their position in CCW order around  $v$  from the parent (or root) edge. A child

of node  $v$  is called a first or second child depending on whether the edge from the child to  $v$  is a first or second edge of  $v$ . A first or second child is *missing* if the first or second edge, respectively, of  $v$  is a non-tree edge. For the root  $s$ , call the first edge searched from  $s$  the first edge, and the other edge the second.

As in Section 2, construct  $M_0^*$  and  $T_0^*$  by splitting each non-tree edge  $e = \langle u, v \rangle$  (with  $u$  the ancestor of  $v$ , say) into two, inserting a degree-two node  $w_e$  into  $T_0^*$  as a first or second child of  $v$  as appropriate. The edge  $\langle w_e, u \rangle$  remains a non-tree edge.

**Lemma 4.1** *In tree  $T_0^*$ , there are five types of internal node:*

1. *A node with an internal node first child and an internal node second child.*
2. *A node with an internal node first child and a missing second child.*
3. *A node with an internal node first child, and a leaf second child.*
4. *A node with a leaf first child and an internal node second child.*
5. *A node with a leaf first child and a leaf second child.*

**Proof:** We show that the remaining four combinations cannot occur. First, no internal node  $v$  of  $T_0^*$  has a leaf first child and a missing second child. For suppose it did. In place of the missing second child is a non-tree edge  $e$ . By construction, the other endpoint of  $e$  is a leaf node  $w_e$  that is a descendent of  $v$ . Node  $w_e$  cannot be a child of  $v$ , however, since otherwise  $e$  would have originally been a loop edge. But by hypothesis  $v$  has no other descendents. This covers one of the four forbidden combinations.

Second, no internal node  $v$  of  $T_0^*$  has a missing first child. For if  $v$  does, then as above, in place of the missing child is a non-tree edge  $e$  whose other endpoint  $w_e$  is a descendent of the second child of  $v$ . Thus, the non-tree edge  $e$  precedes the edge leading to  $w_e$  in CCW order around  $v$ . But this contradicts Lemma 2.1. This covers the remaining three forbidden combinations. ♣

Suppose that we run the labeling procedure of Section 2 on  $M_0^*$  to produce the tree  $M_k^*$ , a copy of  $T_0^*$  with binary labels. Observe that since all internal vertexes have degree three, a non-tree edge can be marked “1” only if it replaces a missing left child. This is forbidden by Lemma 4.1, however, implying that no non-tree edge is marked “1”. In fact, the type of an internal node  $v$  precisely determines how the edges to its descendants are marked: if  $v$  is type-2, then the tree edge to its first child is marked “1” by Step 1. The second edge is a non-tree edge and is deleted. In all other cases both descendent edges are marked “0”.

**Theorem 4.2** *A proper triangulation of  $m$  edges and hence  $r = \frac{2m}{3}$  regions can be encoded in  $(3 + \lg 3)r/2 + O(1) \approx 2.29r$  or  $(3 + \lg 3)m/3 + O(1) \approx 1.53m$  bits.*

**Proof:** Since the edge labels can be inferred exactly from the internal node types, there is no point in saving the label string. In fact, all we need is a pre-order listing of the types of the internal nodes, for this will permit us to reconstruct the entire tree: if leaves are added where indicated, the location at which to place the next internal node in a pre-order traversal of a binary tree is unambiguous.

We shall, then, describe an efficient way of encoding a string of  $r$  internal node types. For technical reasons involving length optimization we choose to do this in two steps: we first distinguish between types 1, 2, and 3-5, then append the information necessary to distinguish the last three from each other. Thus during the first traversal we form  $\sigma_1$  using the following code (“Code A”):

- Type 1: **00**
- Type 2: **01**
- Types 3-5: **1**

We then retrace the tree in the same order, forming a second string  $\sigma'$  over  $\{a, b, c\}$  by writing down, for each node of type 3, 4, or 5, the corresponding letter. This is encoded as a binary string  $\sigma_2$  of length  $|\sigma'| \lg 3 + O(1)$  bits (cf. the proof of Theorem 3.1). The map is encoded by the concatenation  $\sigma = \sigma_1 \circ \sigma_2$ .

Decoding is straightforward: knowing  $r$  we can read off the first  $r$  codewords from Code A, which is prefix-free. We then count the number of nodes of types 3-5, which permits us to read off  $\sigma_2$ , convert it to ternary symbols and use them to determine the types of the “**1**” codewords in  $\sigma_1$ . After reconstructing the dual map, the root node  $s$  is replaced by a single edge between its two children.

How long is  $\sigma$ ? Let  $t_i$  represent the number of nodes of type  $i$ . Observe that since there are  $r/2 + 1$  leaves,  $t_3 + t_4 + 2t_5 = r/2 + 1$ , so  $t_3 + t_4 + t_5 \leq r/2 + 1$ ; since there are  $r$  internal nodes overall, (ignoring the added root  $s$ ) the length of  $\sigma$  is

$$\begin{aligned} |\sigma| &= 2(t_1 + t_2) + (1 + \lg 3)(t_3 + t_4 + t_5) + O(1) \\ &\leq 2(r/2) + (1 + \lg 3)(r/2) + O(1) \end{aligned}$$

bits. This is  $\sim (3 + \lg 3)/2$  bits/region of  $M$ , or  $\sim (3 + \lg 3)/3$  bits/edge. ♣

Our results may be compared with the theoretical limit of  $\approx 1.62$  bits/region or  $\approx 1.08$  bits per edge implied by the enumeration of triangulations due to Tutte [10].

## 5 Remarks

In Sections 2 and 3 we give schemes to encode loop-free maps and stick-free maps in  $\lg 12$  bits per edge, and maps that are both loop-free and stick-free in 3 bits per edge. We leave open, however, the problem of finding an encoding for all planar maps in  $\lg 12$  bits per edge,

the minimum possible. We see no place in our current encoding to store the additional information needed to handle loops and sticks simultaneously. Similarly, although our encoding of triangulations significantly improves the bit requirement over previous schemes, we leave open the problem of finding a minimum length encoding.

## References

- [1] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.
- [2] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21:194–203, March 1975.
- [3] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA., 1972.
- [4] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.
- [5] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17:215–219, 1982.
- [6] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30<sup>th</sup> IEEE Symposium on Foundations of Computer Science, Durham, NC*, October 1989.
- [7] M. Naor. Succinct representations of general unlabeled graphs. *Discrete Applied Mathematics*, 28:303–307, 1990.
- [8] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.
- [9] G. Turàn. Succinct representation of graphs. *Discrete Applied Math*, 8:289–294, 1984.
- [10] W. T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.
- [11] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15(2):249–271, 1963.