Yale University
Department of Computer Science

Short Encodings of Planar Graphs and Maps

Kenneth Keeler[1]
Jeffery Westbrook[2]

YALEU/DCS/TR-871
October 1991

[1]AT&T Bell Laboratories, Holmdel, NJ. 07733.
[2]Department of Computer Science, Yale University, New Haven, CT 06520-2158. Research partially supported by National Science Foundation Grant CCR-8610181.

Abstract

We discuss space-efficient encoding schemes for planar graphs and maps. Our results improve the constant on 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. There are a number of recent results on space-efficient encoding. A standard adjacency list encoding of an unlabeled graph G requires $\Theta m \lg m$ bits, where m and n are the number of edges and vertexes, respectively. Turán [8] gives an encoding of unlabeled connected planar graphs which uses (asymptotically) $4m$ bits[1]. Itai and Rodeh [4] give a scheme for labeled planar graphs requiring $\frac{3}{2} n \lg n + O(n)$ bits, and Naor [6] 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 [5] gives an $\Theta(n)$ space 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.

We shall discuss storage-efficient encodings of unlabeled 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 embeds 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.

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 [5, 8].) The natural measure of map size is the number of edges, and this quantity hence governs the size of our encodings even though 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. Following Tutte [10], we allow graphs and maps to have multiple edges between two vertexes, and to contain loop edges (edges whose endpoints coincide). We show encodings for

- 2-connected maps and graphs in 3 bits per edge,
- arbitrary planar graphs in $\lg 12$ bits per edge,
- arbitrary maps without loop edges in $\lg 12$ bits per edge,

[1] This is not actually stated in [8]; the storage requirement is given as ≤ 12 bits per vertex.

- 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 [10]. There are

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

such, implying that the number of general (unrooted) maps with m edges is $12^m \cdot 2^{-\frac{5}{2}\lg m + O(1)}$. Then given any code for maps with m edges, the number of maps whose codewords are shorter than $m \lg 12 + O(1)$ bits is $o(A_m)$[2]. Another enumeration due to Tutte [9] gives the comparable bound for triangulations, with which we shall compare our result.

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. Our encoding and decoding algorithms, however, 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 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.

2 Encoding 2-connected Maps in 3 Bits/Edge

In this section we describe an encoding scheme for 2-connected maps. To encode a 2-connected graph we embed it using a standard algorithm (e.g. [1]) and then encode the resulting map. An important consequence of 2-connectedness is that the maps contain no loop edges.

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 [7] 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 by choosing a first edge to search out of the the root node. For any vertex u of G and any initial edge $\langle u, v \rangle$, there is a unique TDFS tree T. 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

[2]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.

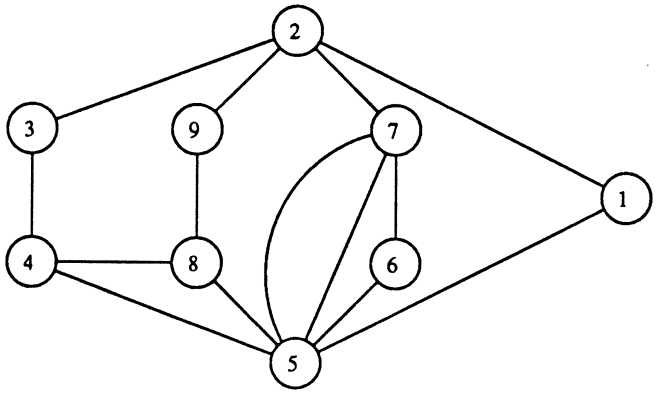

Figure 1: A map M.

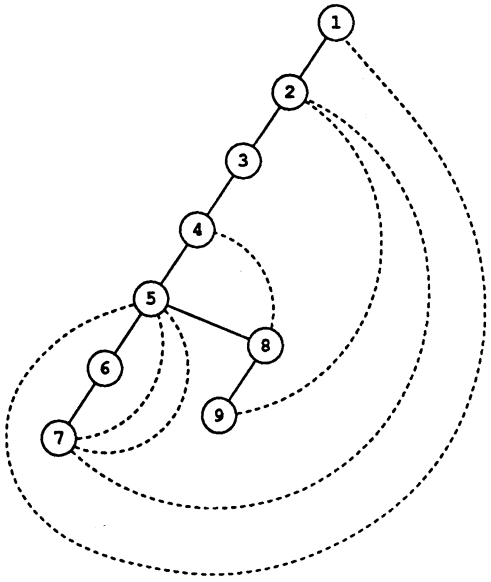

Figure 2: A TDFS of map M. Non-tree edges are shown dashed.

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* [7]. Figure 1 gives an example of a map M and figure 2 shows the TDFS T of M.

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. Then f precedes e in counterclockwise order around u starting at the edge from u to its parent.*

Proof: The edges out of u were explored in counterclockwise order starting at the edge from u to its parent, and 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 2-connected map with n vertexes 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 $\{M_i,\ i = 1, \ldots, k\}$, at each stage possibly deleting an edge and changing some labels. 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 also make w_e a child of v in T_0 by the new tree edge $\langle v, w_e\rangle$ (See Figure 3). 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 $n + k = m + 1$ vertexes and m edges.

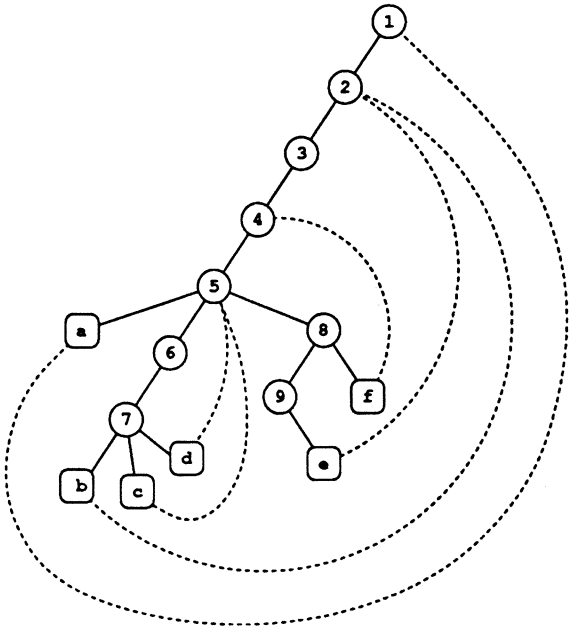

Figure 3: TDFS tree T_0 of M_0.

Since M is 2-connected, every leaf in the TDFS tree T 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.

Let $l_1, l_2, \ldots, l_k$ 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 $e = \langle l_i, u \rangle$. Define the tree path from l_i to u to be $P(i) = (l_i, v = \nu_1, \nu_2, \ldots, \nu_\kappa = u)$. Note that $\kappa \geq 2$, since otherwise $u = v$, implying that e is a loop edge of M, which is forbidden.

The cycle formed by e and $P(i)$ separates the plane into an interior and an exterior region, dividing the remaining vertexes and edges of M_0 into an interior set $INT(i)$ and an exterior set $EXT(i)$ such that $EXT(i)$ contains the root of T_0 and no edge of M connects a vertex of $INT(i)$ with one of $EXT(i)$. Let $f = \langle l_i, v \rangle$ and let g be the edge that immediately precedes e in CCW order around u. Since $u \neq v$, we have $f \neq g$. By Lemma 2.1, edge $\langle \nu_{\kappa-1}, u \rangle$ precedes edge e in CCW order around u. Hence edge g is contained in $INT(i) \cup \{\langle \nu_{\kappa-1}, u \rangle\}$. (See Figure 4.)

Using the above definitions, we process leaf l_i as follows (see Figures 4 and 5).

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

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

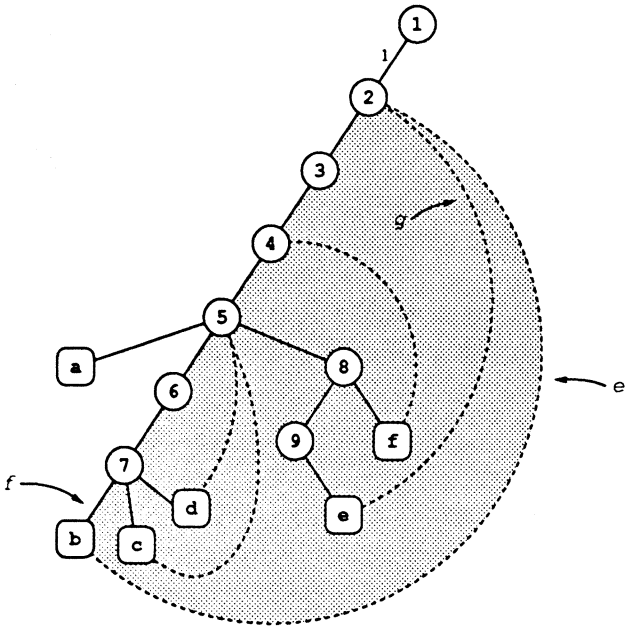

Figure 4: Tree T_1 prior to processing leaf b. (Edges have label 0 unless marked 1.) Shaded area is INT(b).

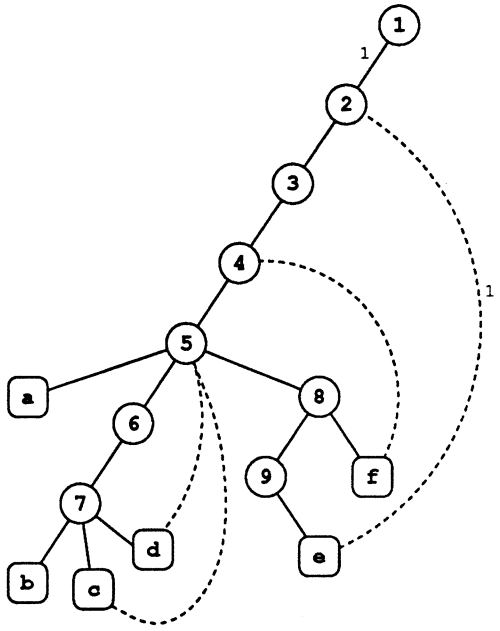

Figure 5: Tree T_2 after processing leaf b.

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, if it is in W, deleting the incident non-tree edge. ♣

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 ν_γ is the least common ancestor of l_i and l_j. Furthermore, since the first edge on the path from ν_γ to l_j must lie inside $INT(i)$, it must follow the edge $\langle \nu_{\gamma-1}, \nu_\gamma\rangle$ in CCW order around ν_γ, 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 satisfied the following: it connected l_i to some ancestor u other than the parent v of l_i; it immediately followed an edge g marked 1; and it enclosed a region in which all edges were marked 0 except for g and possibly the edge f from l_i to its parent. These conditions uniquely determines where the non-tree edge must be inserted. 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 σ'. Simultaneously, we encode the unlabeled tree M_k in a string σ_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 σ_1 and σ_2 to give the encoding of map M.

That this is uniquely decodable is clear: given m, we can read off σ_1 and σ_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 σ_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 Elias's representation of the positive integers [2]. ♣

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 each loop edge so that it is "empty", *i.e.*, its interior consists of a single face. Since a loop edge is a 1-connected component, this is always possible. The embedding is then further rearranged so that for each node v, the incident loop edges and sticks occur last in CCW order around v, starting at the edge from v to its parent in T. Thus, if we walk around v starting at the edge from v to its parent, we encounter first an intermingled collection of regular tree edges and non-tree edges, and then an intermingled sequence of sticks and loop edges with empty interiors. 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. To process l_i, f is labeled $*$, and the non-tree half of the loop edge is deleted, if appropriate. A temporary bit vector is kept that indicates for each edge labeled $*$ whether it led to a loop or a stick. Thus the edge labels are now drawn from the ternary alphabet $\{0, 1, *\}$.

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

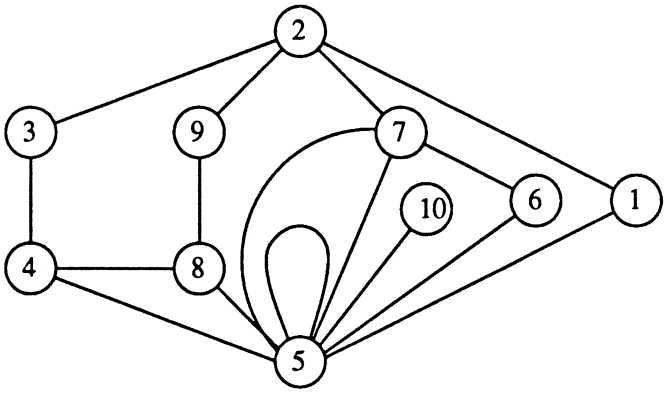

Figure 6: A map M with a stick (vertex 10) and a loop.

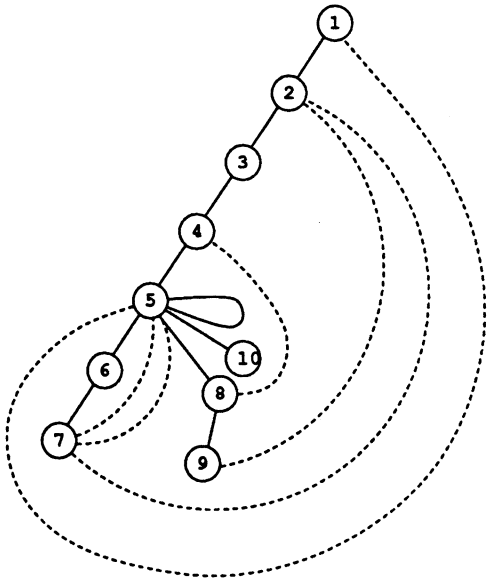

Figure 7: A TDFS of map M after rearrangement of the embedding.

a 1 whenever an edge is descended and a 0 whenever it is subsequently ascended. Note that if an edge leads to a leaf, the descending 1 is immediately followed by the ascending 0. If an edge is labeled $*$ 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 depending on whether the leaf corresponds to a stick or a loop, respectively. 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 descending edge leads to a either a stick leaf or a loop leaf; the bit following the 1 in the tree string indicates which.

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 match the 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: save the original label in a temporary vector and relabel it $*$. Encode the final tree, replacing the redundant ascending 0 for each stick with the stick's original label. ♣

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 the 3-bits-per-edge encoding scheme described in Section 2 can in fact be applied to any map that is free of loops and sticks. 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 dual[3] 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 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 $3r/2$ edges in M and M^*, of which $k = 3r/2 - (r-1) = r/2 + 1$ are non-tree edges.

To proceed: compute a (binary) TDFS tree T^* of M^*. Children of a node v are called "first" or "second," according to their position in CCW order around v from its parent. 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]See [3] for a definition.

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 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.

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. ♣

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 only be marked 1 if it replaces a missing left child. This is forbidden by Lemma 4.1, however, implying that no non-tree edge is ever marked 1. In fact, the type of an internal node v precisely determines how the edges to its descendants are marked:

- If v is a type-1 node, then both the edges from v to its children remain and are marked "0" in M_k^*.
- If v is type-2, then the edge to its first child remains and is marked "1" by Step 1 when the non-tree edge that replaces the second child is deleted.
- If v is type-3, then both edges remain and are marked "0". Neither can be labeled "1" by Step 1 of the labeling procedure. The second edge can only inherit a zero mark in Step 2, since all non-tree edges are marked zero.
- If v is type-4, then both edges remain and are marked "0".
- If v is type-5, then both edges remain and are marked "0".

Theorem 4.2 *A proper triangulation of m edges and $r = \frac{2m}{3}$ regions can be encoded in $(3 + \lg 3)r/2 + O(1) \approx 2.29r$ or $(3 + lg3)m/3 + O(1) \approx 1.53m$ 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 σ_1 using the following code ("Code A"):

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

We then retraverse the tree in the same order, forming a second string σ' over $\{a, b, c\}$ by writing down, for each node of type 3, 4, or 5, the corresponding letter. Let there be l nodes of the three types together (we will show momentarily that $l \leq r/2 + 1$); then as before we encode the ternary string as a binary string σ_2 of length $l \lg 3 + O(1)$ bits. 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 l of nodes of types 3-5, which permits us to read off σ_2, convert it to l ternary symbols and use them to determine the types of the "**1**" codewords in σ_1.

How long is σ? 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, the length of σ 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 $\sim (3 + \lg 3)/2 \approx 2.29$ bits/region of M, or $\sim (3 + lg3)/3 = 1.53$ bits/edge. ♣

Our results may be compared with the theoretical limit of ≈ 1.62 bits/region or ≈ 1.08 bits per edge [9].

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] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17:215–219, 1982.

[5] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30^{th} IEEE Symposium on Foundations of Computer Science, Durham, NC*, October 1989.

[6] M. Naor. Succinct representations of general unlabeled graphs. *Discrete Applied Mathematics*, 28:303–307, 1990.

[7] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[8] G. Turàn. Succinct representation of graphs. *Discrete Applied Math*, 8:289–294, 1984.

[9] W. T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.

[10] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15(2):249–271, 1963.