Compact Routing Tables for Graphs of Bounded Genus

(Extended Abstract)

Cyril Gavoille[1] and Nicolas Hanusse[1]

LaBRI, Université Bordeaux I,
351, cours de la Libération,33405 Talence Cedex, France
`gavoille,hanusse@labri.u-bordeaux.fr`

Abstract. For planar graphs on n nodes we show how to construct in linear time shortest path routing tables that require $8n + o(n)$ bits per node, and $O(\log^{2+\epsilon} n)$ bit-operations per node to extract the route, with constant $\epsilon > 0$. We generalize the result for every graph of bounded crossing-edge number. We also extend our result to any graph of genus bounded by γ, by building shortest path routing tables of $n \log(\gamma+1) + O(n)$ bits per node, and with $O(\log^{2+\epsilon} n)$ bit-operations per node to extract the route. This result is obtained by the use of dominating sets, compact coding of non-crossing partitions, and k-page representation of graphs.

1 Introduction

In point-to-point communication networks or in parallel computers, a routing function is employed in order to deliver messages between processors. As networks grow in size, it becomes important to reduce the amount of memory kept in each node for routing purposes. At the same time, it is essential to route messages along paths that are as short as possible. A *universal routing scheme* is an algorithm which generates a routing function for any given network. One type of trivial universal routing scheme is based on schemes that keep in each node a full *routing table* which specifies an output port for every destination. Though this scheme can guarantee routing along shortest paths, each router has to store locally $n \log d$ bits of memory, where d is the degree of the router (i.e., the number of output ports) and n is the number of nodes in the network. Therefore, this scheme is impractical when dealing with large networks.

In this paper we consider routing table schemes only that generate routing functions that do not change the header of the message along the route, and depend on the destination only. The underlying topology of the network is represented by a graph. All the graphs we consider are simple, i.e., without multi-edges, undirected and connected.

To avoid the implementation of large routing tables, it is possible to code in each node the whole map of the network. However, this *centralized approach* has the following drawback: we need to compute the route with a shortest path algorithm that takes at least a linear time in the number of nodes. So, the time complexity of this approach is exponential since the length of the input is exponential.

A more practical approach to the routing would store a partial view of the topology of the network, and would compute the route in polynomial time with respect to the size of the input, i.e., in time $\log^{O(1)} n$.

Whereas tight bounds exist for the general case [7], no tight bounds exist for the case of planar graphs. Using the centralized approach, $O(n)$ bits are enough, since it is well-known that the number of unlabeled connected planar graphs on n nodes is bounded by c^n, for a constant c such that $\log c < 6.25$ (cf. [4]). The point is that an enumeration formula of a set of objects does not give an efficient way to compute queries fast on individual objects. The time to extract the route is a priori exponential (at least the cardinality of the set, c^n).

A better solution consists of using efficient and compact encoding of planar graphs. Turán proposed in [15] a coding in $12n$ bits computable in linear time. Jacobson's coding [9] allows adjacency queries scanning $O(\log n)$ bits only, but with a $36n$ bit size encoding. Keeler and Westbrook proposed in [10] a more compact encoding in $\log 12$ bits per edge. This leads to a $10.76n$ bit size encoding for dense planar graphs[1], i.e., when $m = 3n - o(n)$. Recently Munro and Raman [13] proposed a coding supporting $O(1)$ time adjacency queries (in the standard integer model) with a $14n+o(n)$ bit size encoding. Finally [3] improved the previous encoding to $4m/3 + 5n + o(n)$ bits, i.e., $9n + o(n)$ bits for dense planar graphs, keeping $O(1)$ time adjacency queries. Note that in the case of planar graphs, shortest paths from one source can be computed in $O(n)$ time [11] instead of $O(n \log n)$ for Dijkstra's algorithm. In the best case this leads to compact routing tables of size $9n + o(n)$ bits with $O(n)$ time to extract the route, but the time complexity is still exponential.

Finally, no shortest path routing scheme is known for planar graphs that uses less than $n \log d$ bits of space ($n \log n$ in the worst-case) and poly-log time.

1.1 Contribution of this Paper

In this paper we show how to route in poly-log time using tables of size at most $8n + o(n)$ bits per node. The same bound holds for non-planar graphs that are planar after removing $o(n/\log n)$ edges. We push even further the optimization of the table to $9n - 8d + o(n)$ bits for nodes of degree d, for $d > n/8$. Moreover the time to build tables is linear for each node, which is optimal. Note that our space bound is smaller than all the known encodings of planar graphs which support *reasonable* time adjacency queries[2] (cf. [3]). Hence our approach is not centralized. See Table 1 for a summary.

The *genus* of a graph G is the smallest integer γ such that G embeds in a surface of genus γ without edge crossings. Planar graphs can be embedded on

[1] This does not mean that the graph is triangulated. Better encodings have been proposed for this particular case.

[2] One can restrict our attention to encodings which support poly-log time adjacency queries.

Routing table size in bits	Time to extract the route	Comments
$n \log n$	$O(\log n)$	Standard routing tables (in bit-operations)
$6.25n$	$2^{n^{O(1)}}$	Information-theoretic upper bound [16]
$9n + o(n)$	$O(n)$	Compact coding of unlabeled planar graphs with $O(1)$ time adjacency queries [3] and fast shortest path algorithm [11]
$8n + o(n)$	$O(\log^{2+\epsilon} n)$, $\epsilon > 0$	Compact routing tables (in bit-operations) [**This paper**]

Table 1. Compact routing tables for planar graphs of n nodes.

a sphere, so $\gamma = 0$. We generalize our result to graphs of genus bounded by γ, $\gamma \geqslant 0$, constructing compact routing tables of $n \log(\gamma + 1) + O(n)$ bits with poly-log time to extract the shortest route. In the worst-case, $\gamma = O(n^2)$ and the size becomes the same order as the standard routing tables, i.e., $O(n \log n)$ bits. This new quantification of the routing information clearly shows that embedding and drawing of graph "help" for the routing. We also prove a lower bound $\Omega(n \log k)$ bits for shortest path routing in graphs of pagenumber k. Because of the lack of space, we refer the reader to [6] for details and proofs.

2 k-page Embedding and Pagenumber

A k-page embedding of a graph consists of a linear ordering of its nodes drawn in a line and of a partition of its edges into k *pages* so that edges residing on the same page do not intersect (see Figure 1 for an example). Such an embedding is sometimes called *book-embedding*. See [2] for a survey. The *pagenumber* of a graph G is the smallest integer k such that G has a k-page embedding. It is NP-hard to compute the pagenumber for general graphs, but there is a linear time algorithm to compute the pagenumber (and its embedding) of planar graphs. Actually the pagenumber of every planar graph does not exceed 4 (cf. [17]).

We show in [6] that $\Omega(kn)$ bits are required in general to code a graph of pagenumber k. This is tight since Jacobson showed in [9] that $9kn$ bits are enough.

3 The Region-Graph and its String Representation

The basic idea of our routing scheme is the following: first we root in each node a minimum spanning tree. Each tree rooted in a node u, say T, represents the

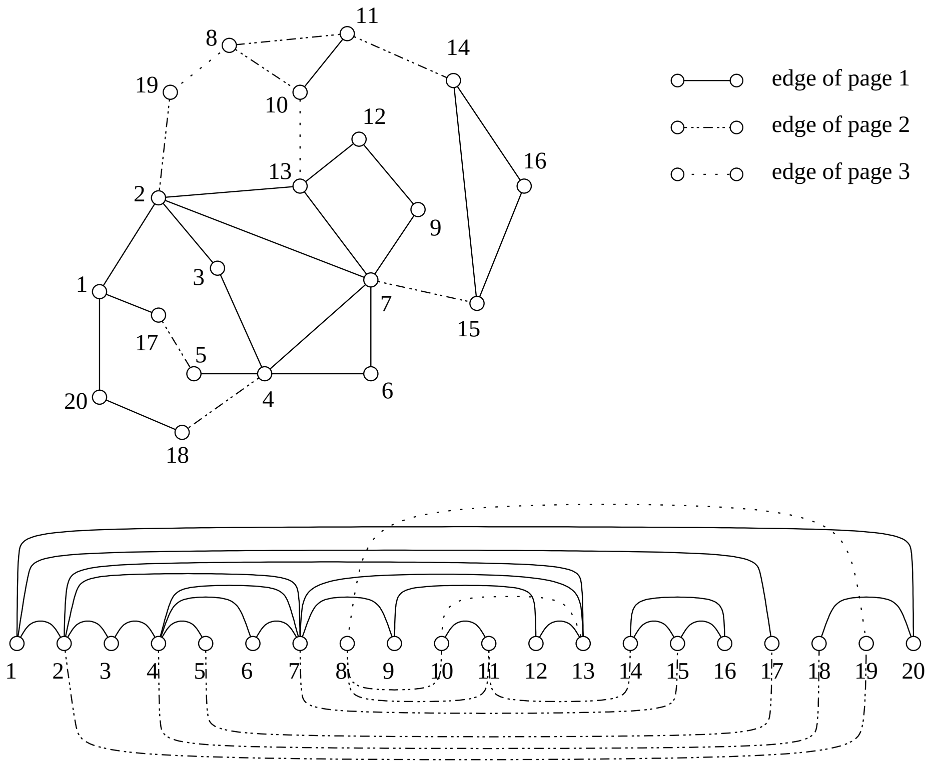

Fig. 1. A planar graph of pagenumber 3 and a 3-page embedding.

routing from u towards all the other nodes in the graph. (Actually our technique works for any tree, not necessary for minimum spanning tree). Roughly speaking, for a given destination v, the routing task in u consists in selecting the unique neighbor of u that is the ancestor of v in T. The destinations having the same ancestor neighbor of u form a *region*. The routing can be implemented by storing in extenso T in u. However, in order to decrease the bit count we store a compact data structure relative to T, called the *region-graph* R_T, that allows to complete our task. In this section we describe how to code such a structure, and in Section 4 we will see how to used it to perform a poly-log time routing algorithm.

3.1 The Region-Graph

Given any graph G, we denote by $V(G)$ its set of nodes, and $E(G)$ its set of edges. Let G be a connected k-page graph with n nodes. We consider a k-page embedding of G. We assume that nodes are labeled with integers taken from $\{1, \ldots, n\}$ linearly ordered by the k-page embedding, and we associate with each edge its unique page p, an integer taken from $\{1, \ldots, k\}$. Let u_0 be a node, let d be the degree of u_0, and let T be an arbitrary spanning tree rooted at u_0. Let us denote by $T - u_0$ the graph obtained from T by removing u_0 and its incident edges. $T - u_0$ is a forest composed of d connected components. Each connected component of $T - u_0$ is a tree called a *region*. We denote by R_i the ith region of $T - u_0$. A region composed of a single node is called an *isolated node*. A region that consists of exactly one edge is called an *isolated edge*. For each tree R_i we choose a root, denoted by $\mathrm{root}(R_i)$, defined as follows: the root of a region R_i which is not an isolated edge is the unique node $r \in V(R_i)$ such that r neighbors u_0. The root of an isolated edge (u, v) is the node $\min\{u, v\}$. We assume that the regions are ordered w.r.t. their root, i.e., $\mathrm{root}(R_1) < \ldots < \mathrm{root}(R_d)$.

Our goal is, given a destination v_0, to find the unique region R_i such that $v_0 \in V(R_i)$. The integer i represents the port of the routing function of u_0. The idea is, starting from v_0, to traverse the region of v_0 (which is a tree) until we reach its roots.

For each page p, $p \in \{1, \ldots, k\}$, let T_p be the subgraph of $T - u_0$ embedded in the pth page of the embedding of G. Formally, $V(T_p) = V \setminus \{u_0\}$, and $E(T_p) = \{(a, b) \mid (a, b) \in E(T - u_0) \text{ and } (a, b) \text{ belongs to the page } p\}$. Note that T_{p_0}, for some p_0, may be empty. Let I_i^p be the set of nodes of the ith connected component of T_p. Given a set I_i^p, the *local-root* w.r.t. I_i^p is the closest node to u_0 in T that belongs to I_i^p. Note that the local-root is unique since nodes of I_i^p induce a subtree of T_p, and therefore a connected subtree of T.

The *region-graph* of T, denoted by R_T, is defined as follows: $V(R_T) = V \setminus \{u_0\}$, and $E(R_T) = \cup_p \cup_i \{(a, b) \mid a, b \in I_i^p, a \neq b, \text{ and } b \text{ is the local-root of } I_i^p\}$. Figure 2 represents a region-graph of a spanning tree T for the example of Figure 1.

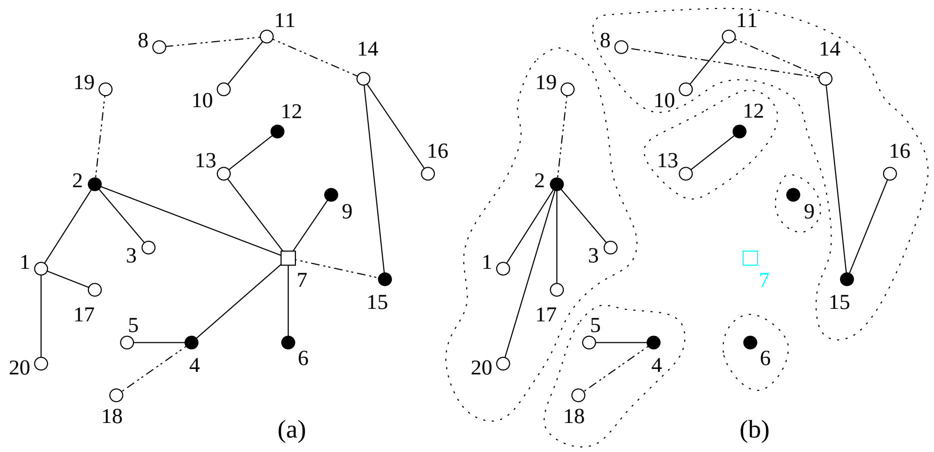

Fig. 2. (a) A spanning tree T rooted in $u_0 = 7$, and (b) its region-graph, R_T. The regions are circled in R_T, and their roots are drawn in black.

Lemma 1. *The region-graph R_T is a forest in which each tree consists of the same set of nodes than the connected component of $T - u_0$.*

According to Lemma 1, and for simplicity, we call each connected component of R_T a *region*.

The main idea in routing along T is to use the region-graph R_T instead of T. The advantage is that to traverse the region of v_0 from v_0 in T we would need an orientation of the edges of the region. That costs n bits of information. As we will see we can recover the father of v_0 in the region-graph without any extra information.

3.2 Coding of the Region-Graph

A partition $P_1 \cup \ldots \cup P_m$ of an ordered set P is a *non-crossing partition* if for all $a, b, c, d \in P$ such that $a < b < c < d$, if $a, c \in P_i$ and $b, d \in P_j$, then $i = j$. Let $\{P_i\}_i$ be a non-crossing partition of P ($\{P_i\}_i$ stands for the set composed of all the P_i's). For each P_i we associate a representative element of P_i, denoted by $\mathrm{cl}(P_i)$. The *representative graph* of $\{P_i\}_i$ w.r.t. the representative function $\mathrm{cl}(\cdot)$ is the graph G_P defined by: $V(G_P) = P$, and $E(G_P) = \cup_i \{(a, b) \mid a, b \in P_i, a \neq b,$ and $b = \mathrm{cl}(P_i)\}$. Every representative graph of any non-crossing partition has a straightforward 1-page embedding. We associate with G_P a nested string of parentheses (called a *balanced* string of parentheses), and defined as follows: for each element $u \in P$ (in order and starting from the minimum), and for each edge $(u, v) \in E(G_P)$, we put a symbol (if $u < v$, and a symbol) if $u > v$.

Lemma 2. *For every $p \in \{1, \ldots, k\}$, $\{I_i^p\}_i$ is a non-crossing partition of $V \setminus \{u_0\}$.*

Therefore, the region-graph R_T is the union, over all pages p, of the representative graph of $\{I_i^p\}_i$ with local-root as representative. This gives also an embedding of R_T in at most k pages.

Let S_p be the string of parentheses associated with page p (more precisely, the string of parentheses of the graph representative of $\{I_i^p\}_i$ with local-root as representative). Let $\mathrm{block}_p(u)$, for every node u, be the sub-string of S_p that corresponds to the incident edges of u drawn on page p. Note that $\mathrm{block}_p(u)$ may be empty.

For every node u of R_T distinct from a root (formally, $u \neq \mathrm{root}(R_i)$ for every $i \in \{1, \ldots, d\}$), we define the *parent-page* of u as the integer p such that the edge $(u, v) \in E(R_T)$ is on the page p, where v is the father of u. The parent-page of a root is not defined.

From R_T we associate a string S_T of multiple types of parentheses defined as follows: (1) in every string $\mathrm{block}_p(u)$ rewrite the symbol (by $(_p$, and the symbol) by $)_p$. (2) For each non-isolated node u, let $\mathrm{block}(u)$ be the concatenation of $\mathrm{block}_p(u)$ over all the pages p in an order such that the block of the parent-page of u (if it exists) is located first. (3) For every isolated node u, set $\mathrm{block}(u) = (_p)_p$, with arbitrary p. (4) Finally, S_T consists of the concatenation of $\mathrm{block}(1)\mathrm{block}(2) \ldots \mathrm{block}(u) \ldots \mathrm{block}(n)$, for every $u \in V(R_T)$.

To complete the coding of S_T we add a bitmap P_T which marks the position of the first symbol of each block. Formally, $P_T[i] = 1$ if and only if $S_T[i]$ starts block(u) for a suitable node u. See the example depicted in Figure 3 corresponding to the region-graph of Figure 2.

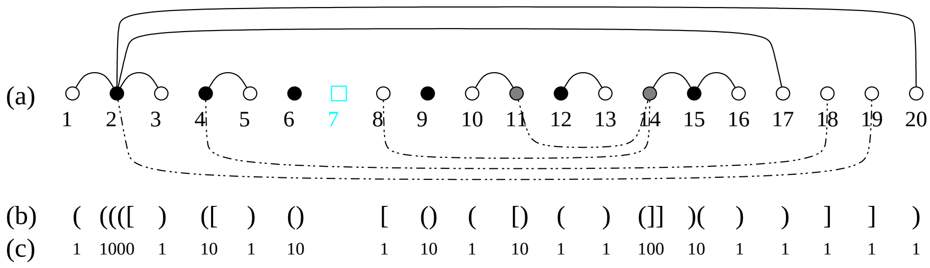

Fig. 3. (a) The 2-page embedding of R_T (local-roots that are not roots are drawn in grey), (b) the string S_T associated with R_T (for simplicity (denotes $(_1$ and [denotes $(_2$, similarly for closing parentheses), and (c) the bitmap P_T.

Lemma 3. *S_T and P_T are computable in linear time, and $|S_T| + |P_T| = (n - 1 - d + i)(4 + 2\log k)$, where i is the number of isolated nodes of R_T.*

4 Compact Routing in the Region-Graph

4.1 A First Routing Algorithm: Route1

In the following algorithm, port(r) denotes a function that returns the port number of the root r. Actually, it corresponds to the number of roots r' such that $r' \leqslant r$, since the roots are ordered.

Algorithm <u>ROUTE1</u>(u_0, v_0)
Input: u_0: the sender, v_0: the destination, $v_0 \neq u_0$
Output: the integer i such that $v_0 \in V(R_i)$

(1) Find the position i of the first symbol of block(v_0) in S_T. Set $u = v_0$.
(2) Find the position j of the matching parentheses of $S_T[i]$.
(3) Find the node u' corresponding to position j.
(4) If $u' = u$, return port(u').
(5) Find the position i' of the first symbol of block(u').
(6) If $i' = j$, let l_u (resp. $l_{u'}$) be the length of the first block of u (resp. u').
  (6a) If $l_u = l_{u'} = 1$, return port$(\min\{u, u'\})$
  (6b) If $l_u > 1$, return port(u), else return port(u')
(7) If $S_T[i']$ and $S_T[j]$ are both in the same page, return port(u').
(8) Set $i = i'$, $u = u'$, and continue at (2).

We consider ω be any function of n not bounded by a constant. So, $n/\omega = o(n)$. W.l.o.g. we assume that $\omega \log n$ is an integer. Let L be the distance between v_0 and the root of its region in R_T.

Lemma 4. *Algorithm* ROUTE1 *runs in* $O(L\omega \log n + \omega^2 \log^2 n)$ *bit-operations with routing tables of* $|S_T| + |P_T| + o(n)$ *bits.*

The implementation and the correctness of ROUTE1 are given [6].

4.2 An $O(\omega \log n)$ steps Routing: Route2

In the time complexity of ROUTE1 (Lemma 4), $L\omega \log n$ becomes the first order term whenever $L > \omega \log n$. Our goal is to bound the number of traversals of the region of v_0, i.e., the number of steps of ROUTE1. For this purpose we select a set of nodes, D_1, satisfying: 1) for every node u at distance greater than $\omega \log n$ from its region's root r in R_T, the path from u to r cuts a node $v \in D_1$ such that the distance[3] between u and v is at most $\omega \log n$ in R_T. 2) D_1 is of size at most $n/(\omega \log n)$.

D_1 can be constructed in linear time using a modified version of Peleg and Upfal's algorithm [14] for the construction of $\omega \log n$-dominating sets: for each region R_{i_0} of height h and of root r we divide the nodes other than r into levels $L_1, \ldots, L_h$ according to their height in R_{i_0} (L_i is the set of nodes at distance i, $i \geqslant 1$). We merge these sets into $\omega \log n$ sets $C_1, \ldots, C_{\omega \log n}$ be taking $C_i = \bigcup_{j \geqslant 0} L_{i + j\omega \log n}$. Each C_i is a $\omega \log n$-dominating set for $V(R_{i_0}) \setminus \{r\}$, and satisfies our property 1). Because these sets form a complete disjoint partition of $V(R_{i_0}) \setminus \{r\}$, at least one set C_{i_0} is of size at most $(|V(R_{i_0})| - 1)/(\omega \log n)$. We merge C_{i_0} with D_1. Finally, the total number nodes in D_1 is at most $(n - 1 - d)/(\omega \log n)$, satisfying our property 2).

Now for each node $u \in D_1$ we assign the port of its root. We store this information in a list D_2. It suffices to insert a Step (1.5) between Step (1) and (2) of ROUTE1 to obtain ROUTE2:

(1.5) If there exists j such that $u = D_1[j]$, return $D_2[j]$.

Clearly, the number of steps for ROUTE2 is bounded by $\omega \log n$.

A *tree-routing family* $\mathcal{F}$ on a graph G is a routing table on G such that for every node the subgraph induced by the set of nodes using the same port is connected. A way to represent a tree-routing family is to use a collection of trees spanning each of these connected subgraphs. So, $\mathcal{F}$ can be seen as a family of spanning trees. So, we can state the main result:

Theorem 1. *Let G be a k-page graph of n nodes, $k \geqslant 1$, and let $\mathcal{F}$ be any tree-routing family on G. $\mathcal{F}$ can be implemented by routing tables of size at most $2n \log k + 4n + o(n)$ bits per node so that the time (per node) to extract the route*

[3] Weights are not taken in account. The distance we are dealing with denotes here the minimum number of edges of a path connecting two nodes.

is $O(\omega^2 \log^2 n)$ bit-operations, where ω is any function as n not bounded by any constant. Moreover, given a k-page embedding of G and given $\mathcal{F}$, the routing tables can be constructed in $O(n)$ time for each node.

We will see later that for unbounded k the bound of Theorem 1 is tight up to a factor 2.

5 Applications and Extensions of the Main Result

Since the tree-routing family can be chosen arbitrarily, it follows that our result can be extended to any weighted graph (weights are non-negative costs assigned to each edge).

[1] shows how to build, for every n, a graph of n nodes, $4n-9$ edges, genus $\gamma \geqslant (n-3)/3$, and pagenumber $k=3$. However, Malitz showed in [12] that for every graph $k = O(\sqrt{\gamma})$, which is tight by [8].

Therefore, from the main theorem we have, choosing $\omega = O(\log^{\epsilon/2} n)$:

Corollary 1. *Every weighted n-node graph of genus γ, $\gamma \geqslant 0$, has shortest path routing tables of $n \log(\gamma+1) + O(n)$ bits per node, and with $O(\log^{2+\epsilon} n)$ bit-operations per node to extract the route, for every constant $\epsilon > 0$.*

Corollary 2. *Every weighted n-node planar graph has shortest path routing tables of $8n + o(n)$ bits per node, and with $O(\log^{2+\epsilon} n)$ bit-operations per node to extract the route, for every constant $\epsilon > 0$. Moreover the table can be constructed in time $O(n)$ per node.*

Furthermore, in [6], we reduce the size of the tables whenever the degree d of the sender is large: $9n - 8d + o(n)$ where $d > n/8$. We also adapt our technics to graphs of crossing-edge number (the smallest number of edges to remove in order to obtain a planar graph) bounded by $o(n/\log n)$. They support shortest path routing tables of size $8n + o(n)$ as well.

At this point we consider the question of whether the bound of Theorem 1 is tight, i.e., if $\Theta(n \log k)$ bits are necessary to route along any tree-routing family. We show that there are tree-routing families on k-page graphs so that $\Omega(n \log k)$ bits are required. Moreover, this lower bound holds for shortest path routing tables. Let $K_{p,q}$ denote the complete bipartite graph on bipartite size p and q respectively.

Theorem 2. *For every n large enough, and for every k, $2 \leqslant k < n/2$, there exists a shortest path tree-routing family $\mathcal{F}$ on $K_{k,n-k}$, an n-node k-page graph, such that any implementation of $\mathcal{F}$ requires at least $(n-2k)\log k - O(1)$ bits simultaneously on at least $n \log k/(4 + 2\log k) - o(n)$ nodes.*

Nevertheless, it is still open to find a lower bound that holds for *any* shortest path tree-routing family, or for any shortest path routing table. To our best knowledge, the only lower bound that holds for any shortest path routing tables is the $\sqrt{n}$-lower bound of [5] for n-node trees.

Acknowledgments: We would like to thank Ondreij Sýkora for his bibliographic support, and Mike Robson for his helpful reading.

References

1. F. Bernhart and P. C. Kainen, *The book thickness of a graph*, Journal of Combinatorial Theory, 27 (1979), pp. 320–331.
2. T. Bilski, *Embedding graphs in books: A survey*, IEE Proceedings-E, 139 (1992), pp. 134–138.
3. R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu, *Compact encodings of planar graphs via canonical orderings and multiple parentheses*, in 25^{th} International Colloquium on Automata, Languages and Programming (ICALP), K. Guldstrand Larsen, S. Skyum, and G. Winskel, eds., vol. 1443 of Lecture Notes in Computer Science, Springer, July 1998, pp. 1–12.
4. A. Denise, M. Vasconcellos, and D. Welsh, *The random planar graph*, Congressus Numerantium, 113 (1996), pp. 61–79.
5. T. Eilam, C. Gavoille, and D. Peleg, *Compact routing schemes with low stretch factor*, in 17^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, ed., Aug. 1998, pp. 11–20.
6. C. Gavoille and N. Hanusse, *Compact routing tables for graphs of bounded genus*, Research Report RR-1213-99, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, Feb. 1999. To appear in ICALP '99.
7. C. Gavoille and S. Pérennès, *Memory requirement for routing in distributed networks*, in 15^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, ed., May 1996, pp. 125–133.
8. L. Heath and S. Istrail, *The pagenumber of genus g graphs is $O(g)$*, in 19^{th} Annual ACM Symposium on Theory of Computing (STOC), 1987, pp. 388–397.
9. G. Jacobson, *Space-efficient static trees and graphs*, in 30^{th} Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Oct. 1989, pp. 549–554.
10. K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Applied Mathematics, 58 (1995), pp. 239–252.
11. P. Klein, S. Rao, M. Rauch, and S. Subramanian, *Faster shortest-path algorithms for planar graphs*, in 26^{th} Annual ACM Symposium on Theory of Computing (STOC), 1994, pp. 27–37.
12. S. M. Malitz, *Genus g graphs have pagenumber $O(\sqrt{g})$*, in 29^{th} Symposium on Foundations of Computer Science (FOCS), IEEE, ed., Oct. 1988, pp. 458–468.
13. J. I. Munro and V. Raman, *Succint representation of balanced parentheses, static trees and planar graphs*, in 38^{rd} Symposium on Foundations of Computer Science (FOCS), IEEE, ed., Oct. 1997, pp. 118–126.
14. D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, Journal of the ACM, 36 (1989), pp. 510–530.
15. G. Turán, *Succint representations of graphs*, Discrete Applied Mathematics, 8 (1984), pp. 289–294.
16. W. T. Tutte, *A census of planar triangulations*, Canadian Journal of Mathematics, 14 (1962), pp. 21–38.
17. M. Yannakakis, *Four pages are necessary and sufficient for planar graphs*, in 18^{th} Annual ACM Symposium on Theory of Computing (STOC), 1986, pp. 104–108.