Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report RR-1213-99

Compact Routing Tables for Graphs of Bounded Genus

by Cyril Gavoille, and Nicolas Hanusse

February, 1999

Compact Routing Tables for Graphs of Bounded Genus*

Cyril Gavoille† Nicolas Hanusse

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, $\gamma \geqslant 0$, 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.

Keywords: Compact routing, routing tables, shortest path, k-page embedding, planar 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.

*A preliminary version of this part of this work appears in the proceedings of the ICALP conference in 1999.

†LaBRI, Université Bordeaux I, 351, cours de la Libération, 33405 Talence Cedex, France. e-mail: {*gavoille,hanusse*}*@labri.u-bordeaux.fr*

A *routing function* R is a distributed algorithm whose role is to deliver messages between nodes of the network. Specifically, the routing function R uses a pair of integer functions (P, H) where P is the *port function* and H is the *header function*. For any two distinct nodes u and v, R produces a path of nodes $(u = u_0, u_1, \ldots, u_k = v)$, a sequence $h_0, h_1, \ldots, h_k$ of headers, and a sequence $p_0, p_1, \ldots, p_k$ of output port numbers. A message with header h_i arriving at node u_i through input port p_i is given a new header $H(u_i, p_i, h_i) = h_{i+1}$, and is forwarded to the output port $P(u_i, p_i, h_i) = p_{i+1}$. Thus, we require that for every $i \in \{1, \ldots, k-1\}$, $H(u_i, p_i, h_i) = h_{i+1}$, $P(u_i, p_i, h_i) = p_{i+1}$ and that p_{i+1} corresponds to the output port number of the link (u_i, u_{i+1}). On each router, the input and output ports numbered 0 are associated with the special link between the router and its host. This allows us to complete the description by imposing the constraints that $p_0 = 0$, $P(u_k, p_k, h_k) = 0$, as well as $h_0 = v$, thus fixing the initial header. Since we are interested in distributed routing schemes, we will express the delivery protocol as a collection of local routing functions $R_u = (P_u, H_u)$, one for each node u. Our goal is to find shortest path routing schemes such that the routing functions generated, R_u, can be coded with low space complexity and that can extract the route in a fast way.

In this paper we consider routing table schemes only, that is schemes that generate routing functions that do not change the header along the route, and depend on the destination only, namely $h_0 = h_1 = ... = h_k = v_0$ (where v_0 is the destination label), and function P does not depend on the input port p_i. Names are in the set $\{1, \ldots, n\}$, and port numbers of a node u in the range $\{1, \ldots, d\}$ where d is the degree of u. The general model is used in [PU89] and in [FG97] to prove lower bounds on the memory size of the routing information needing to be maintained in each router.

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 decrease the size of the routing tables we allow relabeling of the nodes, keeping the range of names in the set $\{1, \ldots, n\}$.

To avoid the implementation of large routing tables, it is possible to code in each node the whole map of the network. This allows small data structure if the underlying graph can be described with a low number of bits. 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. Standard distributed hypotheses assume that local computation is negligible in front of the communication cost. This assertion becomes unreasonable for the centralized approach. Indeed, given the topology of the graph we need $\Omega(m)$ time, where m is the number of edges of the network, to locally compute the shortest route which is always larger than the length of the route. More fundamentally, such routing algorithm computes the routing task in time $\Omega(n)$, where n is the number of nodes of the network, whereas the length of the input (a node address) is on $O(\log n)$ bits. So, the time complexity of this approach is exponential.

The shortest path algorithm is not necessary the bottleneck for the centralized approach. There is a second drawback for networks that support compact representation of their topology. For instance, in [KK96], it is shown that every Cayley graph (a large class of interconnection networks including: hypercube, rings, tori, shuffle-exchange, butterfly, ...) supports shortest path distributed routing algorithms using $O(\log^3 n)$ bits only. This result is based on the representation of Abelian groups, that can be done compactly since a theoretical enumeration formula of such groups shows that only few n-node graphs are Cayley graphs on Abelian groups. However in [KK96] no time bound to extract the route is given. Coding a group can be done with a low number of bits, whereas the time to find a shortest path, i.e., a shortest decomposition of an element in its generators, is quite difficult without the construction of the entire graph. More generally, it is shown in [FG98] that there are some routing functions that can be coded on $O(\log n)$ bits in each router (meaning that the Kolmogorov complexity of the routing function is $O(\log n)$ bits only) such that every implementation of such routing functions on less than $n - o(n)$ bits requires a time and a space to compute the function larger than every constant size stack of powers of 2, i.e., greater than

$$2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}.$$

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 $\log^{O(1)} n$.

Concerning distributed routing schemes, it is shown in [GP96] that $\Omega(n \log d)$ bits are necessary, whereas $n \log d$ are enough by the use of standard routing tables. Precisely, this tight lower bounds means that there are some worst-case graphs on n nodes and maximum degree d, for every $3 \leqslant d \leqslant \epsilon n$ and constant $\epsilon < 1$, such that for every node- and port-labeling, and whatever the shortest path routing function is, the size of any implementation of this function is of size at least $\Omega(n \log d)$ bits. Moreover it is shown that this occurs on $\Theta(n)$ nodes of the graph.

1.1 Planar and Bounded Genus Graphs

Whereas tight bounds exist for the general case [GP96], 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 $c^{n+o(n)}$, for a constant c. So we can theoretically specify any planar graph with an index of $n \log c + o(n)$ bit size. More generally, consider a graph G belonging to any recursive set of graphs of cardinality $c^{n+o(n)}$. (The class of unlabeled planar n-node graphs is a recursive set). To route with $n \log c + o(n)$ bits[1] in a given node u it suffices to store an enumeration algorithm

[1] In all the paper, $\log n$ denotes the logarithm in base two of n.

A of the set of graphs, an index of G in this enumeration, say $\mathrm{id}(G)$, a canonical label of u, ℓ_u, and any shortest path algorithm S, such as Dijkstra's algorithm. The size of the table is bounded by the size of $\mathrm{id}(G)$ ($n \log c$ bits), the size of ℓ_u ($\log n$ bits), and the size of the algorithms A and S which are constant size. The route towards a node v can be extracted in u as follows: 1) using A and $\mathrm{id}(G)$ compute a data structure representing G (for instance an adjacency list). Note that a canonical way to label the nodes (in order to guarantee that each node has the same "view" of the node-labeling) consists of assigning a label i to a node which is the ith created node during this phase. 2) Apply S between u and v to compute the route and the port number. Clearly such "minimal" coding of the routing information implies very large time for local computation of the route.

The best enumeration formula that gives the number of unlabeled planar graphs on n nodes, denoted by $\mathcal{U}(n)$, shows there exists a constant c such that (cf. [DVW96]):

$$\lim_{n \to +\infty} \mathcal{U}(n)^{1/n} \;=\; c \;.$$

This means that asymptotically the number of connected[2] planar graphs tends to $2^{n \log c + o(n)}$. Because there is $O(n)$ ways to embed a 3-connected planar graphs in the plane, by the Tutte's census of planar triangulations [Tut62], we have that $\log c \geqslant \log(256/27) \approx 3.24$. Moreover, it suffices to add at most $3n - 6$ edges to make 3-connected any planar graph. Hence we have that $3.24 < \log c \leqslant \log(256/27) + 3 < 6.25$ (we can specify by a single bit if an edge belongs or not to the original graph). So, asymptotically routing tables of size $6.25n$ bits are enough with the centralized approach in planar graphs.

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, $c^{n+o(n)}$). A better solution consists of using efficient and compact encoding of planar graphs. Turán proposed in [Tur84] a coding in $12n$ bits computable in linear time. Jacobson's coding [Jac89] allows adjacency queries scanning $O(\log n)$ bits only, but with a $36n$ bit size encoding. Keeler and Westbrook proposed in [KW95] a more compact encoding in $\log 12$ bits per edge. This leads to a $10.76n$ bit size encoding for dense planar graphs[3], i.e., when $m = 3n - o(n)$. Recently Munro and Raman [MR97] proposed a coding supporting $O(1)$ time adjacency queries (in the standard integer model) with a $14n + o(n)$ bit size encoding. Finally [CGH+98] 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. All these results imply routing tables of size $O(n)$ bits per node with time in $\Omega(n)$ to extract the route (for each source one needs to run a shortest path algorithm on the whole graph). Note that in the case of planar graphs, shortest paths from one source can be computed in $O(n)$ time [KRRS94] instead of $O(n \log n)$

[2] Actually in [DVW96] it is shown that the number of unlabeled connected or not planar graphs fulfills the same equation for the same constant c.

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

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. Note that the time is still exponential in the size of the input since the destination addresses are on $\log n$ bits.

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 a sphere, so $\gamma = 0$. A p-plane graph of genus γ is a graph of genus γ that embeds in a surface of genus γ so that all the nodes can be covered by at most p disjoint faces. For instance, outerplanar graphs are 1-plane graphs of genus 0.

Using a pure compact routing approach, Frederickson and Janardan showed in [FJ88] that every p-plane graph of genus γ supports shortest path routing tables of *compactness* at most $3p/2 + \gamma$. This means that the routing tables satisfy the property that the set of addresses that use a given directed edge consists of $\lfloor 3p/2 + \gamma \rfloor$ intervals of consecutive integers (modulo n). Routing tables of compactness k are also called *k-interval routing scheme*, and have been introduced by van Leeuwen and Tan [vLT87]. Such a representation implies for p-plane graphs routing tables of size $O(d(p+\gamma)\log n)$ bits per node of degree d with $O(\log n)$ time to extract the route (cf. [Gav98] for a recent survey of this technique). Unfortunately, the parameter p of a graph can be $\Theta(n)$, even for graphs with $\gamma = 0$. Frederickson and Janardan also considered general compact routing schemes for planar graphs with a bound on the average number of bits used of $O(n^{1/3}\log n)$ per node [FJ89]. However, in this seminal work the context is different since their stronger model allows: 1) the use of routing paths of length 3 times the distance (not shortest path); 2) longer names, not taken from $\{1, \ldots, n\}$; 3) changing the header of the messages in some intermediate nodes making the routing paths not necessarily loop-free whereas routing tables induce simple paths. Moreover, their average bound does not avoid the worst-case of $O(n \log n)$ bits on some nodes. 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.

1.2 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[4] (cf. [CGH+98]). Hence our approach is not centralized. See Table 1 for a summary.

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

[4] 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 [Tut62]
$9n + o(n)$	$O(n)$	Compact coding of unlabeled planar graphs with $O(1)$ time adjacency queries [CGH$^+$98] and fast shortest path algorithm [KRRS94]
$8n + o(n)$	$O(\log^{2+\epsilon} n)$, $\epsilon > 0$	Compact routing tables (in bit-operations) [**This paper**]

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

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.

The paper is organized as follows: in Section 2, we define the k-page embedding and the pagenumber of n-node graphs. We show that $\Omega(kn)$ bits are required for any compact encoding of unlabeled and connected graphs of pagenumber k. In Section 3, we introduce a useful structure to code compactly in $O(n \log k)$ bits the routing information of a node: the *region-graph*. It shows that for unbounded k, it is possible to route with less information than the knowledge required for description of the whole graph. In Section 4, we show how to route for each node in poly-log time in the region-graph, and thus in the graph as well. In Section 5, we extend our result to bounded genus graphs, and bounded crossing-edge number. We also prove a lower bound $\Omega(n \log k)$ bits for shortest path routing in graphs of pagenumber k. This lower bound shows that our coding is asymptotically tight up to a factor 2 for unbounded $k = o(n)$. Finally, we conclude with some open problems in Section 6.

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 [Bil92] 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, and this is tight [Yan86]. There are also non planar graphs with pagenumber k, for $k \leqslant 4$. For instance K_5 has pagenumber 3. Nevertheless graphs of pagenumber 1 are outerplanar graphs (including trees), and graphs of pagenumber 2 are subgraphs of Hamiltonian planar graphs (including series-parallel graphs).

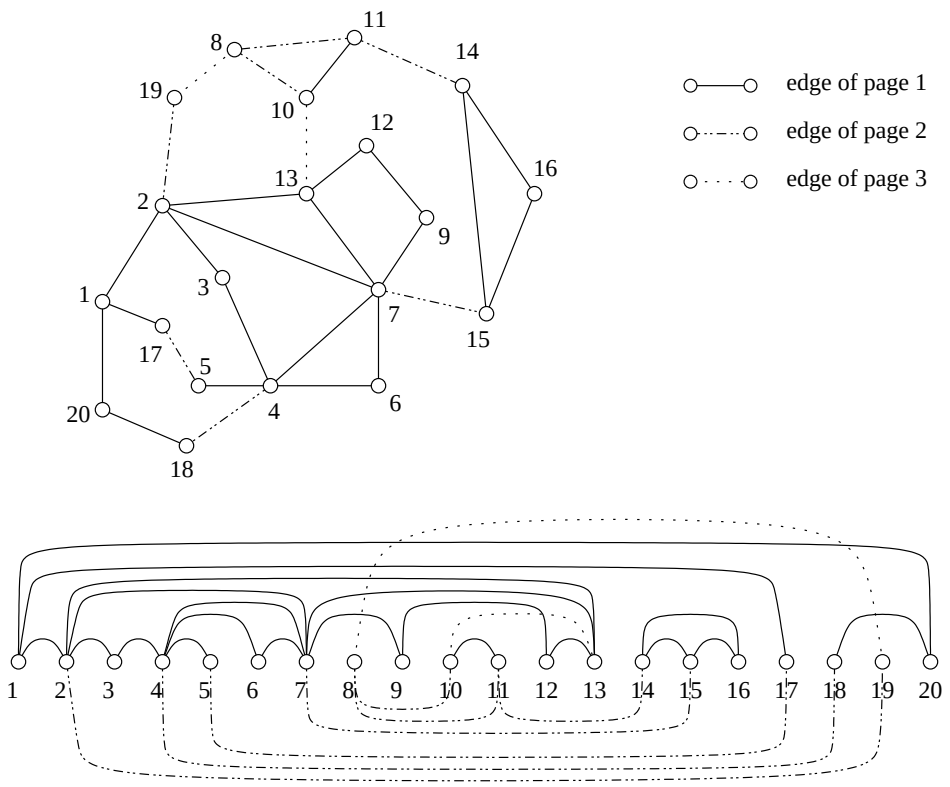

Figure 1: A planar graph of pagenumber 3 and a 3-page embedding.

First, we show that $\Omega(kn)$ bits are required in general to code a graph of pagenumber k. On the other side Jacobson has shown in [Jac89] that $9kn$ bits are enough.

Theorem 1 *The number $\mathcal{U}_{n,k}$ of unlabeled connected graphs of n nodes, and pagenumber k, $1 \leqslant k \leqslant n/2$, satisfies*

$$\mathcal{U}_{n,k} \geqslant \frac{\left(2^k \left(2^k - 1\right)\right)^{n/2-k}}{2k!} .$$

Proof. Here we present a simple proof to show that $\log \mathcal{U}_{n,k} = \Omega(kn)$ for every k. No attempt is made to optimize the constants involved in the calculations. To lower bound $\mathcal{U}_{n,k}$, we construct a family of connected graphs of n nodes and

pagenumber k. For $k = 1$, $\mathcal{U}_{n,1}$ is lower bounded by the number of unlabeled trees (that are 1-page graphs) which is given by the Otter's formula (cf. [Kir90, page 53]):

$$\mathcal{U}_{n,1} \;>\; 0.5349485 \cdot (2.95576)^n \cdot n^{-5/2} \;>\; \frac{2^{n/2-1}}{2 \cdot 1!} \;.$$

Assume $k \geqslant 2$. Note that the pagenumber is maximal for the complete graph of n nodes, K_n. In this case $k = \lceil n/2 \rceil$, for every $n \geqslant 4$, and $k = 1$ otherwise (cf. [Bil92]). Let $\mathcal{M}$ be the set of $k \times (n-2k)$ boolean matrices such that every even column has at least one 0-entry. For each matrix $M \in \mathcal{M}$ we define an n-node graph, G_M, as follows: $V(G_M) = \{1, \ldots, n\}$, and $(i,j) \in E(G_M)$ if: 1) $|i-j| = 1$; 2) $i, j \leqslant 2k$; or 3) $i = 2p-1$, $p \in \{1, \ldots, k\}$, $j > 2k$, and $M_{i,j-2k} = 1$. The formula trivially holds for $n = 2k$, so we assume that $n > 2k$. See Figure 2 for an example with $k = 3$ and $n = 10$.

$$M = \begin{array}{c|cccc} & 5 & 6 & 7 & 8 \\ \hline 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 \end{array}$$

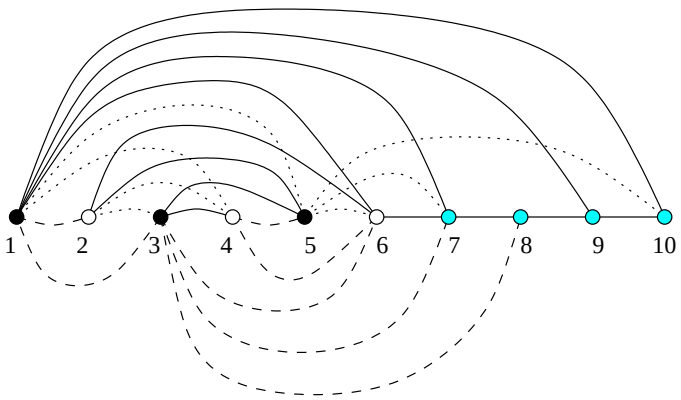

Figure 2: Construction of an n-node graph of pagenumber k.

G_M has a Hamiltonian path $(1,2), \ldots, (i, i+1), \ldots (n-1, n)$, therefore G_M is connected. Since for all $i, j \leqslant 2k$ the edge (i,j) exists, G_M has a clique of size at least $2k$. This implies that its pagenumber is at least the pagenumber of the complete graph K_{2k}, i.e., k since $2k \geqslant 4$. Let us show that G_M has a k-page embedding, and hence that G_M is connected, has n nodes, and has pagenumber k exactly.

Proposition 1 *For every $k \geqslant 1$, K_{2k} has a k-page embedding such that for every page $p \in \{1, \ldots, k\}$ there is no edge (a,b) embedded on the page p with $a < 2p-1$ and $b > 2p-1$.*

Proof. On each page p, $p \in \{1, \ldots, k\}$, we draw the edges: 1) $(i, 2p-i-z)$, for every $i \in \{1, \ldots, p-1\}$ and for every $z \in \{0, 1\}$; 2) $(2p+i-z, 2k-i)$, for every $i \in \{0, \ldots, k-p\}$ and for every $z \in \{0, 1\}$. Remark that there is no edge (a, b) drawn on page p such that $a < 2p-1$ and $b > 2p-1$. Indeed, the edges (a, b) defined by rule 1) satisfy $a < b \leqslant 2p-1$, and edges defined by rule 2) satisfy $2p-1 \leqslant a < b$. Let (a, b), (c, d) be two edges. W.l.o.g. assume that $a < b$, $c < d$, and $a \leqslant c$. Note that (a, b) and (c, d) can be drawn on the same page without any crossing edge if and only if either $a = c$, or $b = d$, or $a < b < c < d$, or $a < c < d < b$. Since there is no crossing edges if $a = c$ or $b = d$, let us assume that $a \neq c$, so $a < c$, and $b \neq d$. Let us show that edges can be drawn with rules 1) and 2) without any crossing edges. Two cases occur: first both edges have been defined by different rules. By the previous remark $a < b \leqslant 2p-1$, and $2p-1 \leqslant c < d$, so $a < b < c < d$. Secondly, both edges are drawn by the same rule. Assume this is rule 1): $(a, b) = (i, 2p-i-z)$ and $(c, d) = (i', 2p-i'-z')$. $a < c$ implies $i < i'$, thus $2p-i > 2p-i'$, and $b = 2p-i-z > d = 2p-i'-z'$, since $z, z' \in \{0, 1\}$, and since $b \neq d$. Hence $a < c < d < b$. For the rule 2): $(a, b) = (2p+i-z, 2k-i)$ and $(c, d) = (2p+i'-z', 2k-i')$. $a < c$ implies $2p+i-z < 2p+i'-z'$, thus $i \leqslant i'$. Hence $b = 2k-i > d = 2k-i'$ ($b = d$ is impossible by assumption), hence $a < c < d < b$.

It remains to show that all the edges of K_{2k} are drawn. One can check that an edge is never defined twice by rule 1) or rule 2). Moreover, the number of edges defined by rule 1) is $2(p-1)$ and the number of edges defined by rule 2) is $2(k-p)-1$ (remark that for $i = k-p$ and $z = 0$ rule 2) defines the edge $(k+p, k+p)$ that should be not considered in our embedding). So, in total we have drawn only once $k(2(p-1)+2(k-p)-1) = k(2k-1)$ edges that is the number of edges of K_{2k}. This completes the proof of Proposition 1. □

To embed G_M, first draw the nodes $1, 2, \ldots, n$ on a straight line, and the edges $(2k, 2k+1), \ldots, (n-1, n)$ on the first page. All the edges of the clique (point 2) of the definition of G_M) are drawn as in Proposition 1. Then, for each page p, $p \in \{1, \ldots, k\}$, we draw the edges $(2p-1, j)$ for every $j \in \{2k+1, \ldots, n\}$ and $M_{p,j-2k} = 1$. This is possible because these new edges are all incident on the same node, $2p-1$, and by Proposition 1 there is no edge of K_{2k} that spans the node $2p-1$. Hence G_M has a k-page embedding.

Let $\mathcal{G}_{n,k}$ be the set of labeled graphs defined from the set $\mathcal{M}$. Counting the number of ways to fill odd and even columns of matrices of $\mathcal{M}$, we have:

$$\begin{aligned} |\mathcal{G}_{n,k}| = |\mathcal{M}| &= \left(2^k\right)^{\lceil (n-2k)/2 \rceil} \left(2^k - 1\right)^{\lfloor (n-2k)/2 \rfloor} \\ &\geqslant \left(2^k \left(2^k - 1\right)\right)^{n/2-k} . \end{aligned}$$

In order to lower bound $\mathcal{U}_{n,k}$, let us calculate the number of nonisomorphic graphs in $\mathcal{G}_{n,k}$. Let $G \in \mathcal{G}_{n,k}$. Let K be a clique of G of maximal size, i.e., with a maximum number of nodes. Note that, by construction G has a clique of size $2k$ (the $2k$-first nodes), and also that the size of the maximal clique is at most $2k$ otherwise the pagenumber of G would be at least $\lceil (2k+1)/2 \rceil > k$.

Let us show that K is unique in G, and thus is composed of the nodes $1, \ldots, 2k$. It suffices to show that every node $u > 2k$ belongs to a clique of size at most $k+1$, and thus cannot belong to K, since for every $k \geqslant 2$, $k+1 < 2k$. Let K_0 be the k odd nodes of $\{1, \ldots, 2k\}$. The j-th column of M corresponds to the node $u = j + 2k$, and defines the connections between u and K_0. Note that there is no edge between $u-1$ and $u+1$, even for $u = 2k+1$. Assume that u is even. In this case, u has degree at most $k+1$ because u neighbors $u-1$, $u+1$, and at most $k-1$ nodes of K_0 since j is even. Thus, u belongs to a clique of size at most $k+2$. However, $u-1$ and $u+1$ are not connected. Therefore, either $u-1$ or $u+1$ does not belong to its clique. So, u belongs to a clique of size at most $k+1$. Now, assume u odd. u is of degree at most $k+2$. u belongs to a clique of size $k+3$ if $u-1$ and $u+1$ both belong to the same clique. If the maximal clique of u contains $u-1$ or $u+1$, its size is at most $k+1$ since $u-1$ and $u+1$ are even, and applying the previous case. Finally, if u contains neither $u-1$ nor $u+1$, the clique is of size at most $k+1$, completing our claim.

There are only two ways of connecting (and drawing) the $n-2k$ nodes not in K, and there are $k!$ ways to draw the k pages and the clique. So, each graph is isomorphic to at most $2 \cdot k!$ graphs in $\mathcal{G}_{n,k}$. Therefore $\mathcal{G}_{n,k}$ has at least $|\mathcal{G}_{n,k}|/(2k!)$ nonisomorphic graphs, completing the proof. □

The coding of k-page graphs of Jacobson [Jac89] in $O(kn)$ bits shows that the lower bound of Theorem 1 is asymptotically tight. Indeed, $\log \mathcal{U}_{n,k} \geqslant kn + o(kn)$, for every $k = o(n)$. [MR97] improved Jacobson's coding by the multiplicative constant for the coding of unlabeled k-page graphs with $2kn+2m$ bits, where m is the number of edges supporting $O(1)$ adjacency queries. This coding is always less than $4kn + 2n$ bits since it is easy to check that $m \leqslant \sum_{i=0}^{k}(n-1-i) < kn + n$ (graphs are assumed to be simple). We can also use the following $2m \log k + 4m$ bit encoding which improves the previous one whenever $m < nk/\log(2k)$.

Theorem 2 *Every unlabeled connected k-page graph of n nodes and m edges can be represented with $2m \log k + 4m$ bits. Moreover, with an overhead of $o(n)$ bits, it is possible to test adjacency in $O(1)$ integer operations.*

Proof. To describe G we use a string of multiple types of parentheses inspired from [CGH+98], and defined as follows: successively for each node u, represent by $(_p$ an edge that belongs to the page p and that connects a node $v > u$. Similarly, we represent by $)_p$ an edge that belongs to the page p and that connects a node $v < u$. The resulting string S has length $2m$ and is composed of $2k$ distinct symbols. Because the drawing of the edges on each page p is planar, the string composed of the symbols $(_p$ and $)_p$ consists of a balanced string of parentheses. The nodes can be located in S by an extra bitmap of $|S| = 2m$ bits. So, in total we use $2m + 2m \log(2k) = 2m \log k + 4m$ bits.

Given two nodes u and v, $u < v$, it is possible from S and the extra bitmap

to rebuild the sub-strings of S associated with u and v respectively denoted by S_u and S_v. u and v are connected if and only if there is a p such that S_u contains a symbol $(_p$, S_v contains symbol $)_p$ (or the reverse if $u > v$), and such that the two parentheses match in S.

Clearly, by efficient implementation of balanced string of multiple types of parentheses as it is done in [CGH+98] we can test adjacency (and some other basic operations) in $O(1)$ integer operations. □

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 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*, that allows to complete our task.

In this section we describe how to code such 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 $E(T_p) = \{(a,b) \mid (a,b) \in E(T - u_0)$ and (a,b) 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 $E(R_T) = \cup_p \cup_i \{(a,b) \mid a, b \in I_i^p, a \neq b,$ and b is the local-root of $I_i^p\}$. Figure 3 represents a region-graph of a spanning tree T for the example of Figure 1.

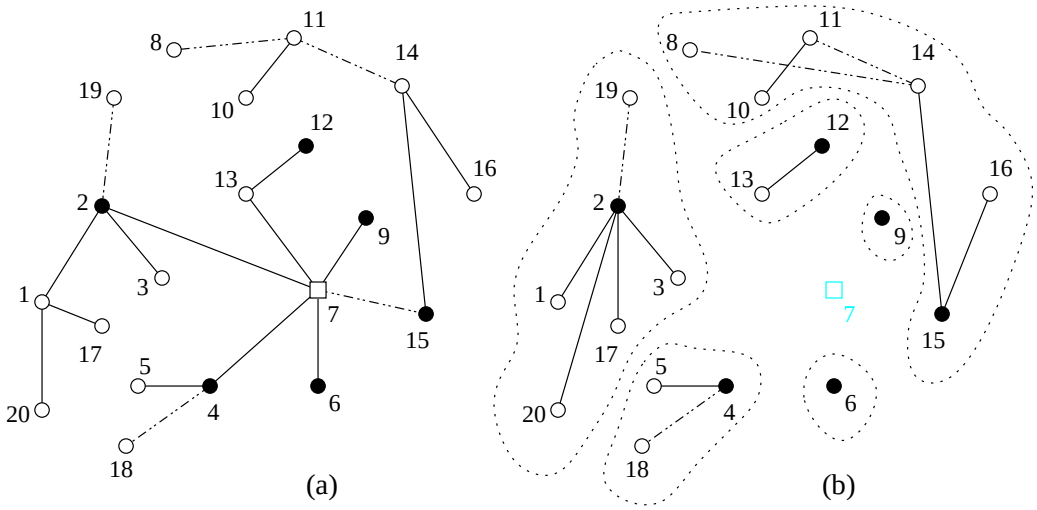

Figure 3: (a) A spanning tree T rooted in $u_0 = 7$, and (b) its region-graph, R_T. The regions are circled in R_T, and their roots are drawn in black.

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 $T - u_0$.*

Proof. I_i^p is the set of nodes of the ith connected component of $T - u_0$ on the page p. The subgraph induced by I_i^p is a subtree of $T - u_0$. The transformation of $T - u_0$ into R_T consists of replacing each induced subtree by another one, precisely by $K_{1,t}$ with $t = |I_i^p|$ and rooted at the local-root w.r.t. I_i^p. Therefore, each connected component of R_T is still a tree with the same set of nodes as $T - u_0$. □

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

Remark. The main idea in routing along T is to use the region-graph instead of T. Indeed, the representation of T might be large (larger than the trivial $2n$ bits representation of unlabeled rooted trees). The nodes of T are already labeled by the k-page embedding of G, and a spanning tree must be designed for all the nodes of G. A mapping of nodes of G into a canonical labeling of the nodes of T could cost an overhead of $n \log n - O(n)$ bits. This represents much more information than our $O(n \log k)$ upper bounds. The most compact coding of labeled trees has a length at least $n \log n - O(\log n)$ bits as well since the number of labeled n-node trees, given by the Cayley formula, is n^{n-2}. Another advantage of using R_T instead of T 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 Non-crossing Partitions and Strings of Parentheses

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 $i = j$. The number of non-crossing partitions is given by the Narayana-numbers

$$\frac{1}{n}\binom{n}{m}\binom{n}{m-1}$$

where $n = |P|$ (cf. [Kir90, page 28]). So, by taking the log of these numbers, to describe an arbitrary non-crossing partition we need of at least $2n - O(\log n)$ bits of information in the worst-case, for $m = n/2$. Non-crossing partitions are in bijection with the paths between the root and the leaves of a rooted plane tree of n edges and m leaves.

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 $E(G_P) = \cup_i \{(a, b) \mid a, b \in P_i, a \neq b, \text{ and } b = \mathrm{cl}(P_i)\}$.

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 $u > v$. See Figure 4 for an example.

Remark. The string of parentheses associated with the representative graph does not represent a coding of the non-crossing partition. For instance, in the string we have no information about the location of single elements of the partition.

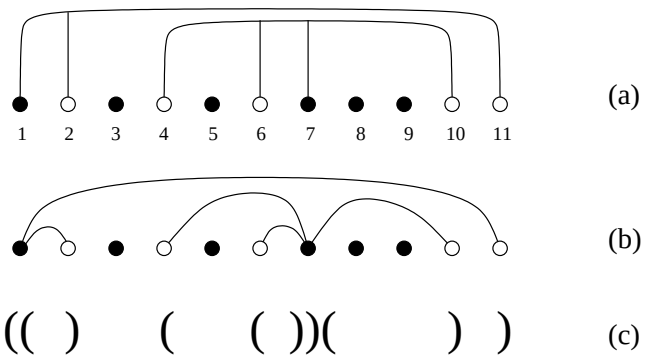

Figure 4: (a) A non-crossing partition, $\{1, 2, 11\} \cup \{3\} \cup \{4, 6, 7, 10\} \cup \{5\} \cup \{8\} \cup \{9\}$, with its representatives drawn in black, (b) the 1-page embedding of its representative graph, and (c) its balanced string of parentheses.

We now introduce some definitions and results about several standard operations on strings.

Let S be a string of l symbols, each symbol belonging to the set $\{1, \ldots, s\}$. S can be seen as a sequence of l integers taken from $\{1, \ldots, s\}$. We denote by $S[i]$ the ith symbol of the string, $i \in \{1, \ldots, l\}$. We denote by $|S|$ the length in bits of S knowing s. We have, $|S| = l \log s$.

- Given a balanced string of multiple types of parentheses S — meaning that every sub-string induced by a given type of parentheses is balanced — and given a position i in S, let $\text{match}_S(i)$ denote the function that returns the position j in S of the unique parenthesis matching with $S[i]$.
- Given a binary string B, let $\text{ones}_B(i)$ denote the function that returns the number of 1's up to and including position i, for $i \in \{1, \ldots, |B|\}$.
- Given a sorted sequence of integers, $A = (a_1, \ldots, a_t)$, $1 \leqslant a_1 \leqslant \ldots \leqslant a_t \leqslant n$, let $\text{rank}_A(a)$ denote the function that, for every integer $a \in \{1, \ldots, n\}$, returns the number of elements of A smaller or equal than a.

Lemma 2

[CGH+98] $\text{match}_S(\cdot)$ *can be implemented with a table of* $|S| + o(|S|)$ *bits, and runs in* $O(1)$ *integer operations. The table consists of coding the string* S *and extra information of* $o(|S|)$ *bits that can be constructed in* $O(|S|)$ *time.*

[BM94] $\text{ones}_B(\cdot)$ *can be implemented with a table of* $|B| + o(|B|)$ *bits, and runs in* $O(1)$ *integer operations. The table consists of coding the bitmap* B *and extra information of* $o(|B|)$ *bits that can be constructed in* $O(|B|)$ *time.*

[Mun96] $\text{rank}_A(\cdot)$ *can be implemented using* $t \log n + o(t \log n)$ *bits, and runs in* $O(1)$ *integer operations. The table consists of coding the sequence* A,

and extra information of $o(t \log n)$ bits that can be constructed in $O(t)$ time.

Remark. The above complexity results are mentioned for the standard integer operation model: each integer operation costs a constant unit of time. We emphasize that actually, in the worst-case, $\log n$ bits need to be read to achieve the constant time results, where $\log n$ is the size of the integers (implementation uses pointers or indexes of size $\log n$). However, a $O(t)$ time complexity in the integer operation model implies a $O(t \log n)$ bit-operations complexity. So, all the functions of Lemma 2 can be implemented with $O(\log n)$ bit-operations.

3.3 Coding of the Region-Graph

Lemma 3 *For every $p \in \{1, \ldots, k\}$, $\{I_i^p\}_i$ is a non-crossing partition of $V \setminus \{u_0\}$.*

Proof. Assume that $I_1^p \cup \ldots \cup I_t^p$ is not a non-crossing partition. Then there exist $a < b < c < d \in V \setminus \{u_0\}$ and a pair i, j, $i \neq j$ such that $a, c \in I_i^p$ and $b, d \in I_j^p$. Since I_i^p and I_j^p are nodes of two disjoint connected components of T_p, a path connecting a, c and a path connecting b, d can be drawn in the same page without crossing edges. This is contradiction with $a < b < c < d$. □

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 $\text{block}_p(u)$, 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 $\text{block}_p(u)$ may be empty.

For every node u of R_T distinct from a root (formally, $u \neq \text{root}(R_i)$ 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 $\text{block}_p(u)$ rewrite the symbol (by $(_p$, and the symbol) by $)_p$. (2) For each non-isolated node u, let $\text{block}(u)$ be the concatenation of $\text{block}_p(u)$ 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 $\text{block}(u) = (_p)_p$, with arbitrary p. (4) Finally, S_T consists of the concatenation of $\text{block}(1)\text{block}(2)\ldots\text{block}(u)\ldots\text{block}(n)$, 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 5 corresponding to the region-graph of Figure 3.

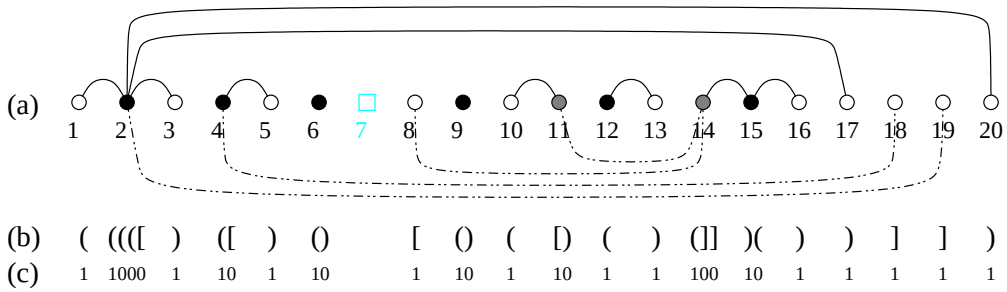

Figure 5: (a) The 2-page embedding of R_T (local-roots that are not roots are drawn in grey), (b) the string S_T associated with R_T (for simplicity (denotes $(_1$ and [denotes $(_2$, similarly for closing parentheses), and (c) the bitmap P_T.

Lemma 4 *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.*

Proof. Given the k-page embedding of G, the tables are computable in linear time. We will show that $|S_T| = 2(n - 1 - d + i)\log(2k)$. Each edge of the region-graph contributes in S_T to a pair of matching parentheses. R_T has d connected components, and $n-1$ nodes. So, R_T has $n-1-d$ edges. Moreover, for each isolated node, we add a pair of parentheses. In total, S_T consists of at most $2(n - 1 - d + i)$ symbols over an alphabet of $2k$ letters (there are at most k types of parentheses). Similarly, $|P_T| = 2(n - 1 - d + i)$. □

4 Compact Routing in the Region-Graph

We give a first version of the routing algorithm in R_T, Route1, we prove its correctness and then its implementation. In a second step, we propose a faster routing algorithm, Route2, that leads to our main result. Finally Route3 improves Route2 only when the degree is large.

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 ROUTE1(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).

4.2 Correctness of Route1

Let us show that after running the steps (2) (3) ... (8), either we have found the root of the connected component of u in R_T, or at Step (8) u' is the father of u in R_T. So, by induction, ROUTE1 returns the root of the region of v_0.

Consider Step (4). If $u' = u$, then u is an isolated node, and by definition u is a root. Consider Step (6). Since $u' \neq u$, $i' \neq i$. Assume $i' = j$. Then u' and u have the same parent-page. So, u or u' is a root. If $l_u = l_{u'} = 1$ then $\{u, u'\}$ is an isolated edge, and therefore the root is by definition the minimum of u and u' (Step (6a)). Otherwise (Step (6b)), the root is the only node whose first block length is greater than 1. Indeed, only one node of a given I_i^p has several incident edges (excepted the case of isolated edges captured by Step (6a)): the local-root of w.r.t. I_i^p. In Step (7), if $S_T[i']$ and $S_T[j]$ are both of the same page then since $i' \neq j$, $l_{u'} > 1$, and therefore u' is a local-root. But, by construction, roots are exactly those local-roots w.r.t. I_i^p such that their first parenthesis (parent-page) is precisely p. So, u' is a root this completes the proof of correctness.

4.3 Implementation of Route1

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.

First, let us break S_T into sub-strings, called *sectors*, each of $\omega \log n$ symbols, labeled successively $1, 2, \ldots, \lceil n/(\omega \log n) \rceil$. Possibly, the last sector could be of length smaller than $\omega \log n$. We construct two lists of integers, S_1 and S_2,

indexed by sectors. For every sector s, $S_1[s]$ contains the node u such that the first parentheses of s belongs to u. $S_2[s]$ contains the number of roots up to sector s (s not included). We introduce some basic functions:

- Given a position i in S_T, node(i) returns the corresponding node. node($\cdot$) can be computed as follows: node(i) = ones$_{P_T}(i) + z$, where $z = 1$ if ones$_{P_T}(i) \geqslant u_0$, and 0 otherwise.
- Given a node u, first(u) returns the position in S_T of the first parentheses of block(u). It can be computed as follows: (1) if $u = 1$ return 1; (2) set $s = \text{rank}_{S_1}(u-1)$, $i = (s-1)\omega \log n + 1$, and $u' = S_1[s]$; (3) while $u' \neq u$ do $i = i + 1$, and if $P_T[i] = 1$, $u' = u' + 1$; (4) return i.
- Given a node u, isroot(u) returns *true* if u is a root, *false* otherwise. isroot($\cdot$) is computed by traversing the father of u, u', and then the father of u', u''. If $u'' = u$, then either u or u' is a root. We conclude that u is a root depending on the length of the first block of u and u' respectively. For that we need to call first($\cdot$), and to check only the two first parentheses in S_T. In the case of an isolated edge (the lengths are both of size 1) we break the tie and return *true* if $u \leqslant u'$.
- Given a node u, port(u) returns the number of roots less than or equal to u. By the ordering of the regions, it corresponds to the port number of u. Using S_2 it can be computed as follows: (1) compute its corresponding sector s using the first position of u in S_T. Set $p = S_2[s]$; (2) starting from the beginning of s and up to the position first(u), we increase p by one for each root encountered with the function isroot($\cdot$). (3) return p.

The lists S_1 and S_2 can be constructed in linear time. Using the complexity results of ones($\cdot$), match($\cdot$) and rank($\cdot$) stated in Lemma 2, we derive:

Lemma 5 *The running time in bit-operations is:* $O(\log n)$ *for* node($\cdot$), $O(\omega \log n)$ *for* first($\cdot$) *and* isroot($\cdot$), *and* $O(\omega^2 \log^2 n)$ *for* port($\cdot$).

Let L be the distance between v_0 and the root of its region in R_T. From the correctness of ROUTE1 and Lemma 5 we have:

Lemma 6 *Algorithm* ROUTE1 *runs in* $O(L\omega \log n + \omega^2 \log^2 n)$ *bit-operations with routing tables of* $|S_T| + |P_T| + |S_1| + |S_2| + o(n)$ *bits.*

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

In the time complexity of ROUTE1 (Lemma 6), $L\omega \log n$ 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[5] 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 [PU89] 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.

A shortest path routing table is not necessary a tree-routing family, but shortest path routing tables can be built on every graph with the use of a minimum spanning tree family. In [dlTNP98], de la Torre, Narayanan, and Peleg pointed out some important properties of tree-routing families: for instance, a shortest path tree-routing family of compactness one (meaning that in the routing table the set of destinations using the same port consists of a single interval of consecutive integers modulo n) supports optimal time and optimal messages distributed broadcast algorithm using only the routing information.

So, we can state the main result:

Theorem 3 *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*

[5]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.

Proof. From Lemma 6, the size of the routing tables used by ROUTE2 in each node u_0 is $|S_T| + |P_T| + |S_1| + |S_2| + |D_1| + |D_2| + o(n)$. By applying Lemma 4, $|S_T| + |P_T| < 2n \log k + 4n$, since the number of isolated nodes is at most the degree of u_0. By construction, $|S_1| = |S_2| = |D_1| = |D_2| = \lceil n/(\omega \log n) \rceil \times \log n = O(n/\omega) = o(n)$. The $\omega \log n$-dominating set guarantees that the number of steps for ROUTE2 is bounded by $\omega \log n$, that completes the proof. □

Note that for $k = 1$, 2, or 3, Theorem 3 does not give the best compact routing. For instance for $k = 1$, i.e., for the case of outerplanar graphs, our upper bound gives $4n + o(n)$ bits, whereas $n + o(n)$ bits are enough (with poly-log time to extract the route) by efficient implementation of interval routing (cf. [Gav98]). However, as we will see later in Paragraph 5, for unbounded k the bound of Theorem 3 is tight up to a factor 2.

4.5 Implementation for Large Degree: Route3

In this paragraph we show how to reduce the size of the tables when the degree of the sender is large. The main idea is the following: we consider the graph obtained from the region-graph R_T by removing all its isolated nodes. Let R'_T be this new graph. Given a destination v_0 distinct from an isolated node, we rename v_0 as v'_0 which corresponds to the rank of the node v_0 for R'_T in the k-page embedding of G. Then, we route applying ROUTE2 on R'_T with input v'_0, and we obtain a port number for R'_T, say p'. Finally, we show how to translate p' into the port number in the original region-graph.

Let us consider the bitmap I_T such that $I_T[u] = 1$ if and only if the node u of R_T is an isolated node. $|I_T| = n - 1$ bits. We construct all the tables needed for ROUTE2 in R'_T. We make an exception for the table D_2 in which we consider the port number of R_T instead of R'_T. This does not change anything about time and space complexities. Let S'_T and P'_T be the two strings used for the routing table of R'_T. By Lemma 4, we have $|S'_T| + |P'_T| < (n-d)(4+2\log k)$.

Given a binary string S, and a position i in S, let $\text{close}_S(i)$ be the function that returns the largest position $j \leqslant i$ such that $S[j] = 0$. It can be easily implemented with a binary search in S: if $\text{ones}_S(i) = \text{ones}_S(i/2)$ we recursively look for j in the range $\{1, \ldots, i/2\}$, and in the range $\{i/2, \ldots, i\}$ otherwise. The routing algorithm is the following:

Algorithm ROUTE3(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) Set $b = \text{ones}_{I_T}(v_0)$.
(2) If $I_T[v_0] = 1$, return $b + \text{port}(\text{close}_{I_T}(v_0))$.
(3) Apply ROUTE2 on $v_0 - b$. Let u' be the last node considered by ROUTE2, and p' be its port number.
(4) If u' is in D_1 return p', and return $p' + \text{ones}_{I_T}(u')$ otherwise.

Complexity: The run time is bounded by Step (2) and Step (3) which both run in $O(\omega^2 \log^2 n)$ bit-operations. Note that $\text{close}_{I_T}(\cdot)$ runs in at most $O(\log^2 n)$ bit-operations, and similarly for Step (4) to check whether $u' \in D_1$. The length of the routing table is at most $|S'_T| + |P'_T| + |I_T| + o(n) < (n-d)(4 + 2\log k) + n + o(n)$ bits. Tables S_1, S_2, D_1, D_2 are of $o(n)$ bit size as stated before.

Correctness: First assume that v_0 is an isolated node ($I_T[v_0] = 1$). Then, it suffices to return the number of nodes that are roots up to v_0. We decompose this value into the number of isolated roots up to v_0 (which are all roots), and the number of roots for the non-isolated node up to v_0. This is achieved by selecting the largest non-isolated node up to v_0 with close$(\cdot)$, and by the use of the function port(x) which by definition returns the number of roots up to x in R'_T. Now, if v_0 is non-isolated, $v'_0 = v_0 - b$ represents the v'_0th non-isolated node of R_T, that is the v'_0th node of R'_T. So, we can apply ROUTE2, and then we correct the port by adding the number of isolated nodes up to the root of v'_0, u'. Note that if u' is a node of the $\omega \log n$-dominating set, we get the port without any translation.

It follows:

Theorem 4 *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 $2(n-d)\log k + 5n - 4d + o(n)$ bits per node of degree d so that the time (per node) to extract the route 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.*

Theorem 4 improves the main theorem (Theorem 3) when $2n\log k + 4n > 2(n-d)\log k + 5n - 4d$, i.e., whenever $d > n/(4 + 2\log k)$. On the other hand, for small degree the use of routing tables is better when $n \log d < 2n \log k + 4n + o(n)$, i.e., whenever $d < 2^4 \cdot k^2$.

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).

5.1 Planar and Genus Bounded Graphs

The *genus* of a graph G is the smallest integer γ such that G embeds in a surface of genus γ without edge crossings. A surface of genus γ is homeomorphic to the sphere with γ crosscaps attached. Planar graphs can be embedded on a sphere, so $\gamma = 0$. Although it is NP-hard to compute the genus of a graph [Tho89], for each fixed γ there is a linear time algorithm to decide whether G is of genus γ [Moh96].

[BK79] 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 [Mal88] that for every graph $k = O(\sqrt{\gamma})$, which is tight by [HI87].

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.*

Proof. It is linear to draw every planar graph G in k pages, where $k \leqslant 4$ is the pagenumber of G (cf. [Bil92]). It is also linear to find a minimum spanning tree rooted in a given node for planar graphs [KRRS94]. We complete the proof by applying Theorem 3. □

Note that if $d > n/8$, by Theorem 4, we can decrease the size of the tables to $9n - 8d + o(n)$ for planar graphs.

5.2 Crossing Edges

In this section we are interested in graphs that are almost k-page graphs, which are graphs drawn into k pages with a limited number of crossing edges (cf. [SSSV98] for a recent survey). Formally, consider an integer $k \geqslant 1$, a graph G, and a subgraph H of G. H is a *k-page excess* for G if the graph obtained

from G by removing all (and only) the edges of H is a k-page graph, i.e., supports a k-page embedding without any crossing edge. A (H,k)-page embedding of G is an embedding in which only the edges of H cross in the k-page drawing of G. We introduce the *k-excess number* of G, denoted by $\mathcal{X}_k(G)$, as the minimum, taken over all k-page excess H of G, of the number of edges of any spanning forest of H. (This number corresponds also to $|V(H)| - \kappa(H)$ where $\kappa(H)$ denotes the number of connected components of H). By definition, for every k and every G, $0 \leqslant \mathcal{X}_k(G) \leqslant n-1$, and $\mathcal{X}_k(G) = 0$ if $k \geqslant k_0$, where k_0 is the pagenumber of G.

The following result generalizes Theorem 3, and improves the result whenever $\mathcal{X}_k(G) = o(n/\log n)$ for some $k < k_0$. Obviously, the next result can be combined with Theorem 4 for large degree nodes.

Theorem 5 *Let G be a graph of n nodes, let $k \geqslant 1$ be an integer, and let $\mathcal{F}$ be any tree-routing family on G. $\mathcal{F}$ can be implemented by routing tables of size at most $2n\log k + 4n + 2\mathcal{X}_k(G)\log n + o(n)$ bits per node so that the time (per node) to extract the route is $O(\omega^2 \log^2 n)$ bit-operations, where ω is any function as n not bounded by any constant.*

Proof. Consider a (H,k)-page embedding of G for suitable H so that any spanning forest of H has $\mathcal{X}_k(G)$ edges. Consider a node u_0 of degree d, and the tree of $\mathcal{F}$ rooted in u_0, T. $T - u_0$ is drawn on at most k pages. Let F be the forest obtained from T by removing all the edges of H. Let $F_0, F_1, \ldots, F_c$ be the $c+1$ connected components that compose F. Let us choose u_i to be any node of F_i, $i \in \{1,\ldots,c\}$. (For convenience, we assume $u_0 \in V(F_0)$.) Each u_i is chosen arbitrary in $V(F_i)$, however if F_i is a single edge, we choose $u_i = \min V(F_i)$. For convenience, we assume $u_0 \in V(F_0)$. Finally, we consider the tree T' of root u_0 obtained from T by adding the edge (u_0, u_i), for every $i \in \{1,\ldots,c\}$.

The idea is to route long T' using the previous compact routing in the region-graph of T', say $R_{T'}$. The routing is valid, except for nodes in F_i, with $i \geqslant 1$. Moreover, the port number is in the range $\{1,\ldots,c+d\}$ instead of $\{1,\ldots,d\}$. Using two lists of c integers, we show how to correct the routing, that is how to simulate the routing of T in $R_{T'}$.

Let U be the list of u_is, $U[i] = u_i$, and let P_U be the list of port numbers in the routing of T for each node of U. So, the route along T from u_0 to any u_i is through the port $P_{U[i]}$, for every $i \in \{1,\ldots,c\}$. As previously done in ROUTE3 for large degree, we construct all the tables needed for ROUTE2 for R'_T. We make an exception for the table D_2 in which we consider the port number of T instead of $R_{T'}$. This does not change anything about time and space complexities.

We consider the following routing algorithm:

Algorithm ROUTE4(u_0, v_0)

(1) If there exists i such that $v_0 = U[i]$, return $P_{U[i]}$.
(2) Apply ROUTE2 on v_0. Let u' be the last node considered by ROUTE2, and p' be its port number.
(3) If u' is in D_1 return p'.
(4) If there exists i such that $u' = U[i]$, return $P_{U[i]}$.
(5) Return $p' - \text{rank}_U(u')$.

Complexity: The run time is bounded by the time of Step (2), $O(\omega^2 \log^2 n)$ bit-operations. Step (1) can be done with a binary search in $O(\log n)$ bit-operations. The same time holds for Step (3-5). Clearly, $|U| + |P_U| = 2c \log n$ bits. Moreover, all the extra tables used for ROUTE4 are $o(n)$ bit size and constructible in linear time. Let us show that $c \leqslant \mathcal{X}_k(G)$, so proving that with an extra information of $2\mathcal{X}_k(G) \log n + o(n)$ bits it is possible to route. The edges of H that are removed of T in order to form F constitute a subgraph of a spanning forest of H, because T is a tree. Each edge we remove from T increases by one the number of connected components, so in total F has exactly 1 plus the number of edges of T that are in H, i.e., $c+1$ connected components. So, $c \leqslant \mathcal{X}_k(G)$.

Correctness: We remark that each u_i is a root of $R_{T'}$, since either they neighbor u_0, or are minimum if they belong to an isolated edge in T'. By the correctness of ROUTE2, we have after Step (2) that u' is either a node of the $\omega \log n$-dominating set, or a root which is either in U or not. If the root is in U we can conclude by looking up P_U, otherwise it suffices to correct the answer by counting the number of roots in $R_{T'}$ that are u_i nodes up to u'. □

Because the pagenumber of planar graphs is bounded by 4, it is interesting to notice that $\mathcal{X}_4(G)$ is a lower bound of the *crossing-edge number*, that is the smallest number of edges to remove from G in order to obtain a planar graph. As a corollary, graphs of crossing-edge number bounded by $o(n/\log n)$ have shortest path routing tables of size $8n + o(n)$ bits.

5.3 A Lower Bound

At this point we consider the question of whether the bound of Theorem 3 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 6 *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.

Proof. $K_{k,n-k}$ has n nodes and a straightforward k-page embedding. The argument we use is based on the incompressibility method in Kolmogorov complexity framework (see [BHV98, GP96] for other examples used for routing complexity). Roughly speaking, we choose a family $\mathcal{F}$ with high Kolmogorov complexity, that is a family that needs a large number of bits to be described or encoded. (This can easily be done by taking $\mathcal{F}$ at random or by a counting argument on the number of ways to choose $\mathcal{F}$). Now, assume that nodes and ports of the graph have been labeled so that the routing table for each node u is optimally encoded in L_u bits. We show that using only the routing tables contained in a suitable subset of nodes, A, we are able to compute a large part of the information contains in $\mathcal{F}$, and so we are able to lower bound L_u for some nodes $u \in A$.

More precisely, consider the set $\mathcal{M}$ of $(n-k)\times(n-k)$ integer matrices where the entries are taken from $\{1,\ldots,k\}$, except for the diagonal entries that are 0. $|\mathcal{M}| = k^{(n-k-1)(n-k)}$. Consider a matrix $M \in \mathcal{M}$ so that any encoding of M has length at least $\log|\mathcal{M}|$ bits. By a counting argument M exists.

Let $A = \{a_1,\ldots,a_{n-k}\}$ be the nodes of the largest partition of $K_{k,n-k}$, and let $B = \{b_1,\ldots,b_k\}$ be the nodes of the other partition. We define $\mathcal{F}$ as follows: for every $b_i \in B$ we choose the unique shortest tree-routing towards the nodes of A, and an arbitrary shortest tree-routing for nodes in $B \setminus \{b_i\}$. For every $a_i \in A$, we choose the unique shortest tree-routing towards the nodes of B, and for every node $a_j \in A \setminus \{a_i\}$, we choose the route through the node b_p where $p = M_{i,j}$. Note that in the latter case, the route through any b_i is a shortest path of length 2. Hence, $\mathcal{F}$ is a shortest path tree-routing family on $K_{k,n-k}$.

As stated before, assume that nodes and ports of each node u of $K_{k,n-k}$ have been labeled so that $\mathcal{F}$ is optimally encoded. Let R_u be the routing function of u, and L_u the length of its coding, for every node u of the graph. Let $F = \{R_u \mid u \in A\}$. Each routing function $R_u \in F$ is an integer function $R_u : \{1,\ldots,n\} \to \{0,1,\ldots,k\}$. F is a collection of finite objects that can be ordered (lexicographically), so $F = \{F_1,\ldots,F_{n-k}\}$ where $F_i \prec F_{i+1}$. Let $L = (l_1,\ldots,l_{n-k})$ be a list of integers such that $F_i = R_{l_i}$. Finally, let $S = \{\sigma_i \mid i \in \{1,\ldots,n-k\}\}$ where each σ_i is a permutation on $\{1,\ldots,k\}$ such that for every $i \neq j \in \{1,\ldots,n-k\}$, $\sigma_i(F_i(l_j)) = R_{l_i}(l_j)$. We consider the following finite process P that builds a matrix M':

For every $i = 1 \ldots n-k$
  For every $j = 1 \ldots n-k$

$$\text{set } M'_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \sigma_i(F_i(l_j)) & \text{otherwise} \end{cases}$$

By construction, $M' = M$. In other words, the knowledge of n, k, F, L, S, and the process P is enough to describe M. Let $C(x)$ denote the Kolmogorov complexity of an object x, i.e., the length in bits of the shortest program (in a fixed language) that prints x and halts (cf. [LV93]). $C(x)$ represents the length of the optimal encoding of x. Thus, by definition, $L_u = C(R_u)$, for every node u. $C(x|i_1, \ldots, i_t)$ denotes the conditional Kolmogorov complexity, that is the length of the shortest program that prints x and halts, with the inputs $i_1, \ldots, i_t$, t fixed. Since we can rebuild M, we have formally:

$$C(F|n,k) + C(L|n,k) + C(S|n,k) + C(P|n,k,F,L,S) \;\geqslant\; C(M|n,k) \;.$$

A list of t integers taken from $\{1, \ldots, m\}$ can be encoded, knowing t and m, by $t \log m$ bits since the number of such lists is m^t. A permutation on $\{1, \ldots, t\}$ can represented by a list of t integers, so by $t \log t$ bits knowing t. So, $C(L|n,k) \leqslant (n-k)\log n$, and $C(S|n,k) \leqslant (n-k)k \log k$. Clearly, $C(P|n,k,F,L,S) = O(1)$. By definition, $C(F|n,k) \leqslant C(F) + O(\log n) \leqslant \sum_{u \in A} L_u$. Recall M satisfies $C(M) \geqslant \log |\mathcal{M}|$, thus $C(M|n,k) \geqslant \log |\mathcal{M}| - O(\log n)$. Therefore,

$$\begin{aligned}
\sum_{u \in A} L_u &\geqslant C(F|n,k) - O(\log n) \\
&\geqslant \log |\mathcal{M}| - (n-k)\log k - (n-k)k \log k - O(\log n) \\
&\geqslant (n-k-1)(n-k)\log k - (k+1)(n-k)\log k - O(\log n) \\
&\geqslant (n-2k-2)(n-k)\log k - O(\log n) \;.
\end{aligned}$$

Hence, there exists $u_0 \in A$ such that:

$$\begin{aligned}
L_{u_0} &\geqslant \frac{\sum_{u \in A} L_u}{|A|} \;\geqslant\; (n-2k-2)\log k - O(1) \\
&\geqslant (n-2k)\log k - O(\log k) \;.
\end{aligned}$$

Actually many nodes require $(n-2k)\log k - O(\log k)$ bits. By Theorem 3, $2n \log k + 4n + o(n) = U$ bits are enough to code $\mathcal{F}$ in all the nodes: in particular, $L_u \leqslant U$, for every $u \in A$. So, by the pigeonhole principle, at least $\sum_{u \in A} L_u / U$ nodes need at least $\sum_{u \in A} L_u / |A|$ bits each, that is:

$$\begin{aligned}
\frac{\sum_{u \in A} L_u}{U} &= \frac{(n-2k-2)(n-k)\log k - O(\log n)}{2n \log k + 4n + o(n)} \\
&\sim n \left(\frac{\log k}{4 + 2 \log k} \right) \;.
\end{aligned}$$

□

For $k = 2$, it follows that $K_{2,n-2}$, which is a series-parallel graph thus a planar graph, needs $n - O(1)$ bits to implement some shortest path routing tables. Moreover this occurs on $n/6 - o(n)$ nodes. For unbounded $k = o(n)$, the lower bound gives $n \log k - O(n)$, and the upper bound $2n \log k + O(n)$. So, Theorem 3 cannot be improved by a factor 2. Hence, for planar graphs

(constant k), the only lower bound we can derive is $n - O(1)$ bits simultaneously for $n/6 - o(n)$ 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 [EGP98] for n-node trees. (Actually it is stated not only for shortest path but for any routing tables.)

6 Conclusion, Open Problem, and Future Work

We have shown that routing tables of size $n \log(\gamma + 1) + O(n)$ bits are enough to route in poly-log time along shortest paths for every n-node graph of genus γ. The problem of finding a tight lower bound is still open. In particular for planar graphs ($\gamma = 0$), we wonder if $\Omega(n)$ bits are required, in particular for the class of 1-page graphs which corresponds to the class of outerplanar graphs. The best lower bound known for compact routing tables in planar graphs is due to [EGP98] with $c\sqrt{n}$ bits, $c \approx 3.70$.

Another direction for future work is to consider a tree-routing family of *stretch factor* s. That is a routing table that corresponds to a routing along an s-spanner which is a tree (an s-spanner if a subgraph H of the original graph G such that the distance in H between two nodes is bounded by s times the distance in G). Shortest path routing tables are tree-routing families of stretch factor exactly 1. We show that any tree-routing family of a k-page graph can be implemented with at most $O(n \log k')$ bits with poly-log time to extract the route, where $k' \leqslant k$ is the number of pages used by each s-spanner. Intuitively, k' decreases with increasing s. It would be interesting to find a trade-off between s and k'.

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

References

[BHV98] H. Buhrman, J.-H. Hoepman, and P. Vitányi, *Space-efficient routing tables for almost all networks and the incompressibility method*, SIAM Journal of Computing, (1998). To appear.

[Bil92] T. Bilski, *Embedding graphs in books: A survey*, IEE Proceedings-E, 139 (1992), pp. 134–138.

[BK79] F. Bernhart and P. C. Kainen, *The book thickness of a graph*, Journal of Combinatorial Theory, 27 (1979), pp. 320–331.

[BM94] A. Brodnik and J. I. Munro, *Membership in constant time and minimum space*, in European Symposium on Algorithms (ESA),

vol. 855 of Lectures Notes in Computer Science, Springer-Verlag, 1994, pp. 72–81.

[CGH+98] 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.

[dlTNP98] P. de la Torre, L. Narayanan, and D. Peleg, *Thy neighbor's interval is greener: A proposal for exploiting interval routing schemes*, in 5^{th} International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, 1998.

[DVW96] A. Denise, M. Vasconcellos, and D. Welsh, *The random planar graph*, Congressus Numerantium, 113 (1996), pp. 61–79.

[EGP98] 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., August 1998, pp. 11–20.

[FG97] P. Fraigniaud and C. Gavoille, *Universal routing schemes*, Journal of Distributed Computing, 10 (1997), pp. 65–78.

[FG98] P. Fraigniaud and C. Gavoille, *A theoretical model for routing complexity*, in 5^{th} International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, July 1998, pp. 98–113.

[FJ88] G. N. Frederickson and R. Janardan, *Designing networks with compact routing tables*, Algorithmica, 3 (1988), pp. 171–190.

[FJ89] G. N. Frederickson and R. Janardan, *Efficient message routing in planar networks*, SIAM Journal on Computing, 18 (1989), pp. 843–857.

[Gav98] C. Gavoille, *A survey on interval routing*, Theoretical Computer Science, (1998). To appear.

[GP96] 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.

[HI87] 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.

[Jac89] G. Jacobson, *Space-efficient static trees and graphs*, in 30^{th} Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, October 1989, pp. 549–554.

[Kir90] P. Kirschenhofer, *Enumeration of trees*, in Séminaire Lotharingien de Combinatoire, 25^{e} Session, A. R. Kräuter, ed., September 1990, pp. 19–61.

[KK96] E. Kranakis and D. Krizanc, *Boolean routing on Cayley networks*, in 3^{rd} International Colloquium on Structural Information & Communication Complexity (SIROCCO), N. Santoro and P. Spirakis, eds., Carleton University Press, June 1996, pp. 119–124.

[KRRS94] 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.

[KW95] K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Applied Mathematics, 58 (1995), pp. 239–252.

[LV93] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, 1993.

[Mal88] S. M. Malitz, *Genus g graphs have pagenumber* $O(\sqrt{g})$, in 29^{th} Symposium on Foundations of Computer Science (FOCS), IEEE, ed., October 1988, pp. 458–468.

[Moh96] B. Mohar, *Embedding graphs in an arbitrary surface in linear time*, in 28^{th} Annual ACM Symposium on Theory of Computing (STOC), 1996, pp. 392–397.

[MR97] 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., October 1997, pp. 118–126.

[Mun96] J. I. Munro, *Tables*, in 16^{th} FST&TCS, vol. 1180 of Lectures Notes in Computer Science, Springer-Verlag, 1996, pp. 37–42.

[PU89] D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, Journal of the ACM, 36 (1989), pp. 510–530.

[SSSV98] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo, *Crossing numbers of graphs, lower bound techniques and algorithms: A survey*, Theoretical Computer Science, (1998). To appear. Appears also as Graph Drawing '94, LNCS 894, pp. 131-142.

[Tho89] C. Thomassen, *The graph genus problem is NP-complete*, Journal of Algorithms, 10 (1989), pp. 568–576.

[Tur84] G. Turán, *Succint representations of graphs*, Discrete Applied Mathematics, 8 (1984), pp. 289–294.

[Tut62] W. T. Tutte, *A census of planar triangulations*, Canadian Journal of Mathematics, 14 (1962), pp. 21–38.

[vLT87] J. van Leeuwen and R. B. Tan, *Interval routing*, The Computer Journal, 30 (1987), pp. 298–307.

[Yan86] 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.

Contents

1 Introduction **1**

1.1 Planar and Bounded Genus Graphs 3

1.2 Contribution of this Paper 5

2 k-page Embedding and Pagenumber **6**

3 The Region-Graph and its String Representation **11**

3.1 The Region-Graph 11

3.2 Non-crossing Partitions and Strings of Parentheses 13

3.3 Coding of the Region-Graph 15

4 Compact Routing in the Region-Graph **16**

4.1 A First Routing Algorithm: Route1 16

4.2 Correctness of Route1 17

4.3 Implementation of Route1 17

4.4 An $O(\omega \log n)$ steps Routing: Route2 18

4.5 Implementation for Large Degree: Route3 20

5 Applications and Extensions of the Main Result **22**

5.1 Planar and Genus Bounded Graphs 22

5.2 Crossing Edges 22

5.3 A Lower Bound 24

6 Conclusion, Open Problem, and Future Work **27**