# Compact Routing Tables for Graphs of Bounded Genus[*]

Cyril Gavoille[†] Nicolas Hanusse

June 16, 2000

**Abstract**

This paper deals with compact shortest path routing tables on weighted graphs with $n$ nodes. For planar graphs 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, for any constant $\epsilon > 0$. We obtain the same bounds for graphs of crossing-edge number bounded by $o(n/\log n)$, and we generalize for graphs of genus bounded by $\gamma > 0$ yielding a size of $n \log \gamma + O(n)$ bits per node. Actually we prove a sharp upper bound of $2n \log k + O(n)$ for graphs of pagenumber $k$, and a lower bound of $n \log k - o(n \log k)$ bits. These results are obtained by the use of dominating sets, compact coding of non-crossing partitions, and $k$-page representation of graphs.

**Keywords:** compact data structures, routing tables, shortest path, planar graphs, bounded genus, $k$-page embedding

## 1 Introduction

In point-to-point communication networks a routing function is employed in order to deliver messages. 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.

### 1.1 The Model

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 $u = u_0, u_1, \ldots, u_k = v$, a sequence $h_0, h_1, \ldots, h_k$ of headers, a sequence $p_0, p_1, \ldots, p_k$ of

[*]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*

output port numbers, and a sequence $q_0, q_1, \dots, q_k$ of input port numbers. A message with header $h_i$ arriving at node $u_i$ through input port $q_i$ is given a new header $H(u_i, q_i, h_i) = h_{i+1}$, and is forwarded on the output port $P(u_i, q_i, h_i) = p_i$. Thus, we require that for every $i \in \{0, \dots, k-1\}$, $H(u_i, q_i, h_i) = h_{i+1}$, $P(u_i, q_i, h_i) = p_i$ and that the link $(u_i, u_{i+1})$ has output port number $p_i$ at $u_i$, and input port number $q_{i+1}$ at $u_{i+1}$ (in- and output port number of a same link may differ). 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 $q_0 = p_k = 0$, 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, $R_u$'s, can be coded with low space complexity (a short algorithm), and such that the route can be extracted 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 $h_0 = h_1 = ... = h_k = v$ (where $v$ is the destination label), and the function $P$ does not depend on the input port $q_i$. Labels of nodes are in the set $\{1, \dots, n\}$, and port numbers of a node $u$ in the range $\{1, \dots, d\}$ 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 weighted graph (weights are non-negative costs assigned to each edge). 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 to choose all the labels (nodes and ports) in advance, keeping the range of $\{1, \dots, n\}$ for nodes and $\{1, \dots, d\}$ for ports. If a model prevents from relabeling, the space complexity for a shortest path routing scheme is $\Theta(n)$ bits per node for a ring and $\Theta(n \log n)$ bits per node for the complete graph (cf. [FG97]). Clearly, for such simple topologies, the complexity drops to $O(\log n)$ bits per node after relabeling.

## 1.2 Centralized vs. Distributed Approach

To avoid the implementation of large routing tables, we can 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, even if it is not always doable, this *centralized approach* has several drawbacks. First, one need to compute the route with a shortest path algorithm in the router. That takes at least a linear time in the number of nodes. Standard distributed hypotheses assume that local computation is insignificant 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 centralized approach has 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: hypercubes, rings, tori, shuffle-exchange, butterflies, ...) supports shortest path distributed routing algorithms using $O(\log^3 n)$ bits only. This result is based on the representation of groups that can be done compactly since a theoretical enumeration formula of such groups shows that only few $n$-node graphs are Cayley graphs. However, no polynomial time bound to extract the route is known. 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 implemented with a program[1] not longer than $O(\log n)$ bits in each router. But every implementation of such routing functions stored in less than $n - o(n)$ bits requires an amount of time and space to compute the function larger than every constant size stack of exponentials, 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 of the routing function, 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 in the worst-case (whereas $n \log d$ are enough). Precisely, this tight lower bounds means that there are some $n$-node graphs of maximum degree $d$, for every $3 \leqslant d \leqslant \epsilon n$ and constant $\epsilon < 1$, such that for every node and port labeling[2], 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. However, the average case is better, since [BHV99] showed that allmost all (unweighted) graphs have shortest path routing tables of size $\Theta(n)$ bits per node.

## 1.3 Planar and Bounded Genus Graphs

Whereas tight bounds exist for general graphs [GP96], no tight bounds exist for planar graphs, even for the unweighted case. 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 assymptotically $c^{n+o(n)}$, for a constant $c$. The exact value of $c$ is still unkown, but from [Tut62] one can derive that[3] $3.24 < \log c < 6.25$. So we can theoretically specify any planar graph with an index of $6.25n$ bit size. More generally, consider a graph $G$ belonging to any recursive family of graphs of cardinality $2^{N+o(N)}$, $N > \log n$. To route with $N+o(N)$ bits in a given node $u$ we have to store an enumeration algorithm $\mathcal{A}$ of the family, an index of $G$ in this enumeration, say $\mathrm{id}(G)$, a canonical label of $u$, say $\ell(u)$, and any shortest path algorithm $\mathcal{S}$ such as Dijkstra's algorithm. The size of the table is bounded by the size of $\mathrm{id}(G)$ ($N + o(N)$ bits), the size of $\ell(u)$ ($\lceil \log n \rceil$ bits), and the size of the algorithms $\mathcal{A}$ and $\mathcal{S}$ (constant size). The route towards a node labeled $\ell(v)$ can be extracted in $u$ as follows: 1) using $\mathcal{A}$ and $\mathrm{id}(G)$ compute[4] a data structure for $\mathcal{S}$ representing $G$; 2) Apply $\mathcal{S}$ on $G$ between $\ell(u)$ and $\ell(v)$ to compute a shortest route, and return the first edge (or a canonical output ports number) on this path.

So, asymptotically routing tables of size $6.25n$ bits are enough with the centralized approach in planar graphs. Clearly such "minimal" coding of the routing information implies very large time for local computation of the route. Moreover it applies only for unweighted graphs.

[1]The result holds for any fixed programming language, say C or FORTRAN.

[2]Ranges of labels are $\{1, \ldots, n\}$ for nodes and $\{1, \ldots, d\}$ for ports.

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

[4]A canonical way to label the nodes of $G$ (and the ports number as well) in order to guarantee consistency in the labeling consists of assigning the label $i$ to a node which is the $i$th created node during the generating phase of $G$ by $\mathcal{A}$.

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. A priori, the time to extract the $i$th object from its set might be the cardinality of the set, larger than $2^n$ for unlabeled planar graphs. 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[5], 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 still 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]. 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 of the routing function since the destination addresses are on $\log n$ bits.

The *genus* of a graph $G$ is the smallest integer $\gamma$ such that $G$ embeds in a surface of genus $\gamma$ without edge crossings. Planar graphs can be embedded on a sphere, so $\gamma = 0$. A $p$-plane graph of genus $\gamma$ is a graph of genus $\gamma$ that embeds in a surface of genus $\gamma$ 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 $\gamma$ 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. [Gav99] for a recent survey of the interval routing 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: $O(n^{1/3} \log n)$ per node [FJ89]. However, in this seminal work the context is different since their 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 and bounded genus graphs that uses less than $n \log n$ bits of space in each node and poly-log time.

## 1.4 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 for weighted planar graphs. The same bound holds for non-planar graphs that become 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

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

for each node. Note that our space bound is smaller than all the known encodings of planar graphs which support "reasonable" time adjacency queries[6]. Our approach is not centralized. See Table 1 for a summary.

| Routing table size in bits | Time to extract the route | Method & comments |
|---|---|---|
| $n \log n$ | $O(\log n)$ | Standard routing tables (in bit-operations), weighted |
| $6.25n$ | $2^{n^{O(1)}}$ | Information-theoretic upper bound [Tut62], unweighted |
| $9n + o(n)$ | $O(n)$ | Compact coding of unlabeled planar graphs with $O(1)$ time adjacency queries [CGH$^+$98] and fast shortest path algorithm [KRRS94], unweighted |
| $8n + o(n)$ | $O(\log^{2+\epsilon} n)$ | Compact routing tables [**This paper**] (in bit-operations), weighted |

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

By the use of $k$-page embedding of graphs, we generalize our result to weighted graphs of genus $\gamma > 0$. We construct compact routing tables of $n \log \gamma + O(n)$ 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.

The paper is organized as follows: in Section 2, we define $k$-page embedding of a graph, and we introduce a dedicated compact data structure that allows to represent all the routing information at each node with only $O(n \log k)$ bits: the *region-graph*. In Section 3, we show how to use the region-graph to route in poly-log time. In Section 4, we extend our result to bounded genus and bounded crossing-edge number graphs. In Section 5, we prove a lower bound of $\Omega(n \log k)$ bits for shortest path routing in graphs of pagenumber $k$, and we conlude by some remarks.

## 2 $k$-page Embedding and the Region-Graph

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 $k \leqslant 4$. For instance $K_5$ has pagenumber 3. Nevertheless graphs of pagenumber 1 are

[6]Restricting our attention to encodings which support poly-log time adjacency queries.

outerplanar graphs (including trees), and graphs of pagenumber 2 are subgraphs of Hamiltonian planar graphs (including series-parallel graphs).

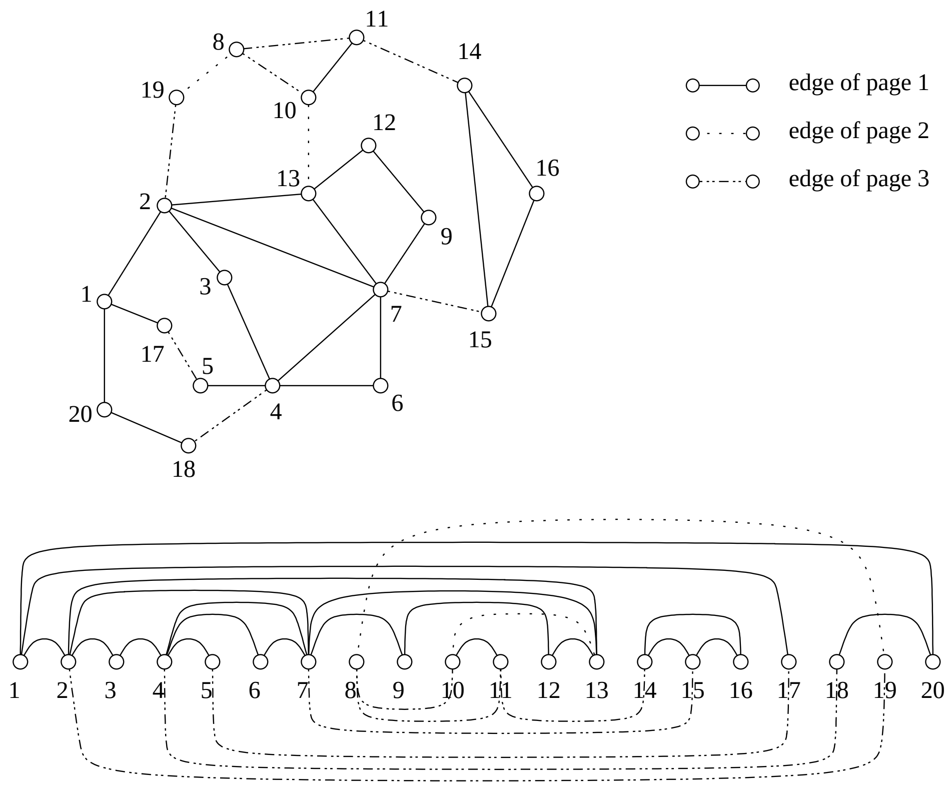

Figure 1: A planar graph of pagenumber 3 and a 3-page embedding.

The sketch of our routing scheme is the following: we choose the node labeling according to the $k$-page embedding of the graph. Then 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 spanning 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 an ancestor of $v$ in $T$. The routing table could be implemented by storing in extenso $T$ in $u$. However, in order to decrease the bit count (and the search time) we store a compact data structure relative to $T$, called the *region-graph*, that allows to perform our task.

## 2.1 The Region-Graph

Let $G$ be an $n$-node connected $k$-page graph with node set $V(G)$ and edge set $E(G)$. We consider a $k$-page embedding of $G$. We assume that $V(G) = \{1, \ldots, n\}$ such that nodes are linearly ordered by the $k$-page embedding of $G$. 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 $u_0$ (for instance a minimum spanning tree if $G$ is weighted). Once $T$ has been fixed, $T$ is considered as an unweighted graph. 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 $i$th 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 $R_i$ 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 $\min\{u, v\}$ (recall that nodes are integers in $\{1, \ldots, n\}$). 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 index $i$ will represent the output port number returned by the routing function at $u_0$ for $v_0$.

For each page $p \in \{1, \ldots, k\}$ let $T_p$ be the subgraph of $T - u_0$ embedded in the $p$th 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 $i$th 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 (unweighted) $T$ 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 the graph 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 2 represents a spanning tree $T$ and a region-graph for $u_0 = 7$ for the example of Figure 1.

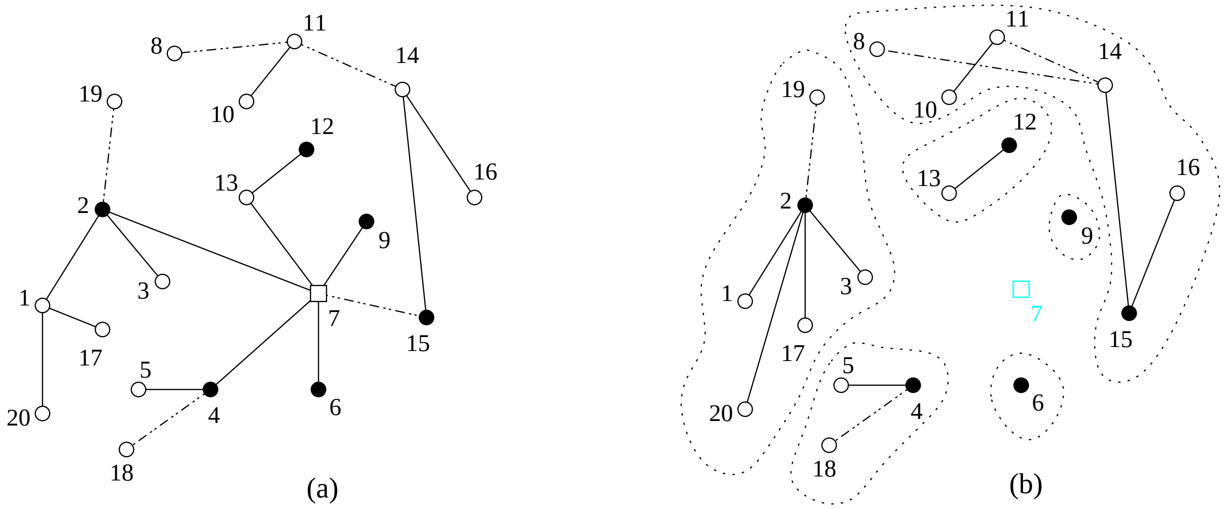

Figure 2: (a) A spanning tree $T$ rooted in $u_0 = 7$, and (b) its region-graph, $R_T$. The regions are circled in $R_T$, and their roots are drawn in black. Note that the root of a region is not always a neighbor of $u_0$ (e.g., node 12), and that edges of the region-graph may cross w.r.t. the initial node embedding of $G$.

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

**Proof.** $I_i^p$ is the set of nodes of the $i$th 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 the complete bipartite graph $K_{1,t-1}$ where $t = |I_i^p|$ 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*. We emphaze that to find $i$ such that $v_0 \in V(R_i)$ we can use $R_T$ instead of $T$. This will allow us to save bits, $T$ being a labeled tree[7]. For this search we traverse the data structure $(R_T)$ following the edges of $R_T$ from $v_0$ up to the root of the region of $v_0$. Then, knowing the node $r = \mathrm{root}(R_i)$ we show how to guess $i$, the output port number to route towards $v_0$. For concreteness, if $v_0 = 8$, our walk into $R_T$ will be $8, 14, 15$. Then, being able to detect whether a node $x \in \{1, \ldots, n\}$ is a root or not, we can stop the walk and count the number of roots less than $r$ to find $i$, the roots being ordered (hence for $v_0 = 8$, we return 6). The goal of the following paragraphs is to show that this can be done with a compact representation of $R_T$ (and other small precomputed tables) allowing to complete our task in $O(\log^{2+\epsilon} n)$ total time.

## 2.2 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 $i = j$. 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 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 3 for an example. Note that in the string of parentheses we have no information about the location of single elements of the partition.

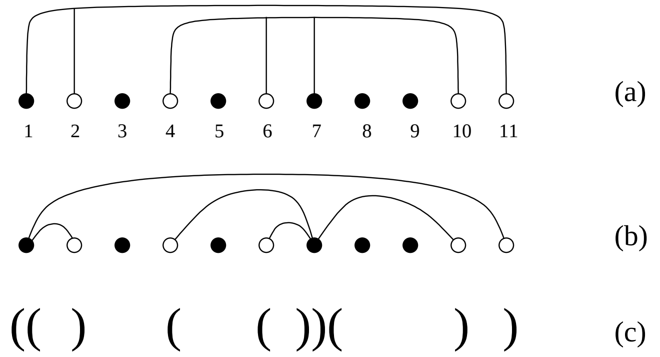

Figure 3: (a) A non-crossing partition, $\{1,2,11\} \cup \{3\} \cup \{4,6,7,10\} \cup \{5\} \cup \{8\} \cup \{9\}$, with its representatives drawn in black, (b) the 1-page embedding of its representative graph, and (c) its balanced string of parentheses.

We now introduce some definitions and results about several standard operations on strings. Let $S$ be a string of $\ell$ symbols, each symbol being represented by an integer taken from $\{1, \ldots, s\}$. We denote by $S[i]$ the $i$th symbol of the string, $i \in \{1, \ldots, \ell\}$. We denote by $|S|$ the length in bits of $S$ knowing $s$. We have, $|S| = \ell \log s$.

- Given a balanced string of multiple type parentheses $S$ — meaning that every sub-string induced by a given type of parenthesis is balanced — and given a position $i$ in $S$, let $\mathrm{match}_S(i)$

[7]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 nodes of $T$ costs an overhead of $\Theta(n \log n)$ bits.

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**

**[MR97, CGH$^+$98]** *If* $|S| = n$, $\text{match}_S(\cdot)$ *can be implemented with a table of* $n + o(n)$ *bits, and runs in* $O(1)$ *integer operations. The table consists of coding the string* $S$ *and extra information of* $o(n)$ *bits that can be constructed in* $O(n)$ *time.*

**[Mun96, BM99]** *If* $|B| = n$, $\text{ones}_B(\cdot)$ *can be implemented with a table of* $n + o(n)$ *bits, and runs in* $O(1)$ *integer operations. The table consists of coding the bitmap* $B$ *and extra information of* $o(n)$ *bits that can be constructed in* $O(n)$ *time.*

**[FKS84]** $\text{rank}_A(\cdot)$ *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 operation on integers of size $O(\log n)$ bits 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. However, an $O(t)$ time complexity in the integer operation model implies an $O(t \log n)$ bit-operations complexity. So, all the functions of Lemma 2 can be implemented with $O(\log n)$ bit-operations.

## 2.3 Encoding 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 $a < b < c < d \in V \setminus \{u_0\}$ such that $a, c \in I_i^p$ and $b, d \in I_j^p$, and $i \neq j$. 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, i.e., $u \neq \text{root}(R_i)$ 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 type 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 $R_T$ 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 $\text{block}(u)$ for a suitable node $u$. See the example depicted in Figure 4 corresponding to the region-graph of Figure 2.

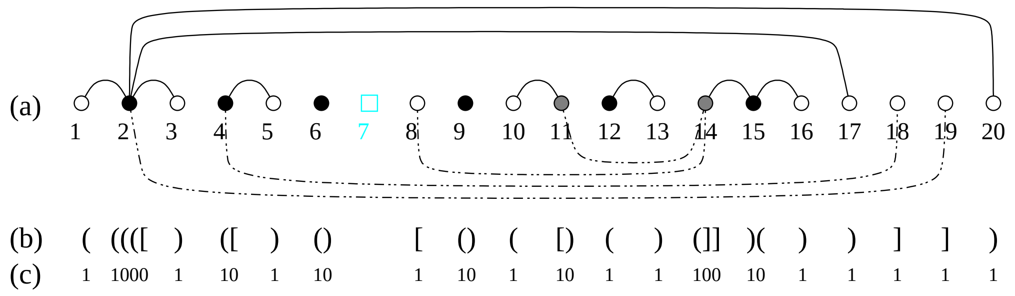

Figure 4: (a) The 2-page embedding of $R_T$ (local-roots that are not roots are drawn in grey), (b) the string $S_T$ associated with $R_T$ (for simplicity ( denotes $(_1$ and [ denotes $(_2$, similarly for closing parentheses), and (c) the bitmap $P_T$.

**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, $|P_T| = 2(n-1-d+i)$. ▮

## 3 Compact Routing with the Region-Graph

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

### 3.1 A First Search Algorithm: Route1

In the following algorithm, $\text{port}(r)$ 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 number of symbols 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).
```

## 3.2 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. $I_i^p$. At 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. Finally, if Step (8) is encountered, $u'$ is a father of $u$, completing the proof of correctness.

## 3.3 Implementation of Route1

We consider $\omega$ 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 parenthesis 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($i$) $= \text{ones}_{P_T}(i) + z$, where $z := 1$ if $\text{ones}_{P_T}(i) \geqslant u_0$, and 0 otherwise.

- Given a node $u$, first($u$) returns the position in $S_T$ of the first parenthesis 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 $P_T[i] = 1$, set $u' := u' + 1$; (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.*

### 3.4 An $O(\omega \log n)$ steps Search: 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[8] 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, say $C_{j_0}$, is of size at most $(|V(R_{i_0})| - 1)/(\omega \log n)$. We merge $C_{j_0}$ 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 output port number 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:

[8]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.

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

Note that find $j$ in Step (1.5) can be performed in $O(\log n)$ bit-operation by a binary search in a sorted list (sorting $D_1$ costs $O(n/\omega) = o(n)$, a sub-linear time of precomputation). Now, 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.

So, we can state the main result:

**Theorem 1** *Let $G$ be a $k$-page graph of $n$ nodes, 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 such that the time (per node) to extract the route is $O(\omega^2 \log^2 n)$ bit-operations, where $\omega$ is any unbounded function in $n$. Moreover, given $\mathcal{F}$ and a $k$-page embedding of $G$, 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, $|S_1| = |S_2| = |D_1| = |D_2| \leqslant \lceil n/(\omega \log n) \rceil \cdot \lceil \log n \rceil = O(n/\omega) = o(n)$ bits. 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 small values of $k$, Theorem 1 does not give the best compact routing scheme. 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 still a poly-log time to extract the route) by efficient implementation of interval routing (cf. [Gav99]). However, as we will see later in Section 5, for unbounded $k$ the bound of Theorem 1 is optimal up to a factor 2.

### 3.5 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. We treat isolated nodes in a different way.

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 $\mathrm{close}_S(i)$ 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 $\mathrm{ones}_S(i) = \mathrm{ones}_S(i/2)$ 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 $b := \mathrm{ones}_{I_T}(v_0)$.
 (2) If $I_T[v_0] = 1$, return $b + \mathrm{port}(\mathrm{close}_{I_T}(v_0))$.
 (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 $p' + \mathrm{ones}_{I_T}(u')$ 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 $\mathrm{close}_{I_T}(\cdot)$ runs in at most $O(\log^2 n)$ bit-operations, and similarly for Step (4) to check whether $u' \in D_1$. The size 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 nodes 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 $\mathrm{close}(\cdot)$, say $x$, and by the use of the function $\mathrm{port}(x)$ 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 $v'_0$th non-isolated node of $R_T$, that is the $v'_0$th node of $R'_T$. So, we can apply ROUTE2, and then we correct the port number 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 number without any translation.

It follows:

**Theorem 2** *Let $G$ be a $k$-page graph of $n$ nodes, 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 $d$ such that the time (per node) to extract the route is $O(\omega^2 \log^2 n)$ bit-operations, where $\omega$ is any unbounded function in $n$. Moreover, given $\mathcal{F}$ and a $k$-page embedding of $G$, the routing tables can be constructed in $O(n)$ time for each node.*

Theorem 2 improves the main theorem (Theorem 1) 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 $d < 2^4 \cdot k^2$.

## 4 Applications and Extensions of the Main Result

### 4.1 Planar and Genus Bounded Graphs

The *genus* of a graph $G$ is the smallest integer $\gamma$ such that $G$ embeds in a surface of genus $\gamma$ without edge crossings. A surface of genus $\gamma$ is homeomorphic to the sphere with $\gamma$ cross-caps 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 $\gamma$ there is a linear time algorithm to decide whether $G$ is of genus $\gamma$ [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 $\gamma > 0$ has shortest path routing tables of $n \log \gamma + O(n)$ 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 1. ▮

Note that if $d > n/8$, by Theorem 2, we can decrease the size of the tables to $9n - 8d + o(n)$ for planar graphs.

### 4.2 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. As we will see this extends the notion of crossing-edge number. Formally, consider an integer $k > 0$, 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 and only if $k \geqslant k_0$, where $k_0$ is the pagenumber of $G$.

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 (cf. [SSSV98] for a recent survey). The following result generalizes Theorem 1, and improves the result whenever $\mathcal{X}_k(G) = o(n/\log n)$ for some $k < k_0$. And 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. Obviously, the next result can be combined with Theorem 2 for large degree nodes.

**Theorem 3** *Let $G$ be a graph of $n$ nodes, let $k > 0$ 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 $O(\log^{2+\epsilon} n)$ bit-operations, for every constant $\epsilon > 0$.*

**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 $F_i$, $i \in \{1,\ldots,c\}$. Each $u_i$ is chosen arbitrary in $V(F_i)$ (but preferably choose a neighbor of $u_0$), but if $F_i$ is a single edge, we choose $u_i = \min V(F_i)$ (as all the nodes are ordered with respect to the linear ordering induced by the $(H,k)$-page embedding of $G$). 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 along $T'$ using the previous compact routing in the region-graph of $T'$, say $R_{T'}$. By construction, $T' - u_0$ has a $k$-page embedding, and thus $R_{T'}$ too. The search is valid, except for node destinations in $F_i$ for some $i \geqslant 1$ as there are more regions in $R_{T'}$ than in $R_T$. Moreover, the port number is possibly in the range $\{1,\ldots,c+d\}$ instead of $\{1,\ldots,d\}$. Using two lists of $c$ integers of $\{1,\ldots,n\}$, we show how to correct the routing, that is how to simulate the search in $T$ in $R_{T'}$.

Let $U$ be the list of $u_i$'s, $U[i] = u_i$, 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, we construct all the tables needed in 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 search 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 $p' - \mathrm{rank}_U(u')$.
```

*Complexity:* Let us choose $\omega = O(\log^{\epsilon/2} n)$. The run time is bounded by the time of Step (2), $O(\omega^2 \log^2 n) = O(\log^{2+\epsilon} n)$ bit-operations. Step (1) can be done with a binary search in $O(\log n)$ bit-operations. The same time holds for steps (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 we can to make a search. 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 it neighbors $u_0$, or $u_i$ is minimum if it belongs to an isolated edge in $T'$. By the correctness of ROUTE2, we have after Step (2) of ROUTE4 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 $u'$. ▮

## 5 A Lower Bound

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 a tree-routing family. We show that there are tree-routing families on $k$-page graphs such that $\Omega(n \log k)$ bits are required. Moreover, this lower bound holds in the particular case of shortest path routing tables on unweighted graphs. Let $K_{p,q}$ denote the complete bipartite graph on bipartite size $p$ and $q$ respectively.

**Theorem 4** *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 $(n-2k)\log k - 3\log n$ bits for a node.*

**Proof.** We use a counting argument based on the number of shortest path routing tables needed to satisfy the set of all the possible shortest path tree-routing families on $K_{k,n-k}$. Actually this proof can be reformulated using the incompressibility method in Kolmogorov complexity framework (see [BHV99, GP96] for other examples used for routing complexity lower bounds).

$K_{k,n-k}$ is an $n$-node $k$-page graph since it has a straightforward $k$-page embedding. Let $A = \{a_1, \ldots, a_{n-k}\}$ be the set of 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. Let $V = A \cup B$. Consider the set $\mathcal{M}$ of all $(n-k) \times (n-k)$ integer matrices where the entries are taken from $\{1, \ldots, k\}$, except for the diagonal entries that are null. Note that $|\mathcal{M}| = k^{(n-k)(n-k-1)}$.

To each matrix $M \in \mathcal{M}$, we associate a tree-routing family $\mathcal{F}^M$ built as follows: for every $b_i \in B$ we choose an arbitrary (but fixed) shortest tree-routing towards all the nodes, i.e., $V \setminus \{b_i\}$. For every $a_i \in A$, we choose the unique shortest tree-routing towards the nodes of $B$, and towards every node $a_j \in A \setminus \{a_i\}$, we choose the route through the node $b_\alpha \in B$ such that $\alpha = M_{i,j}$. Note that in the latter case, the set of routes towards $V \setminus \{a_i\}$ form a shortest tree-routing. Hence, for every $M$, $\mathcal{F}^M$ is a shortest path tree-routing family on $K_{k,n-k}$.

Let $\mathcal{U} = \bigcup_{M \in \mathcal{M}} \mathcal{F}^M$. For every $\mathcal{F}^M \in \mathcal{U}$, let $\mathcal{F}^M_u$ denote the tree-routing rooted at $u$. Note that $|\mathcal{U}| = |\mathcal{M}|$.

Consider any routing scheme $\mathcal{S}$ on $K_{k,n-k}$. I.e., for every tree-routing family $\mathcal{F}^M \in \mathcal{U}$ on $K_{k,n-k}$, $\mathcal{S}$ must return a node labeling $\ell^M$, a port labeling $p^M$, and a routing table $R^M$ that implements $\mathcal{F}^M$. More precisely, for every node $u \in V$:

1. $\ell^M(u) \in \{1, \ldots, n\}$ is the label of $u$;
2. $p^M(u,v) \in \{1, \ldots, \deg(u)\}$, for $v$ neighbor of $u$, is the port number of the edge $(u,v)$;
3. $R^M(u,d)$, $d \in \{1, \ldots, n\}$, returns $p^M(u,v)$, the port of the edge $(u,v)$ such that $(u,v)$ is the edge of $\mathcal{F}^M_u$ leading to the destination labeled $d$.

We denote by $R^M_u$ the function $R^M(u,\cdot)$. Let $L_R(M) = (R^M_{a_1}, \ldots, R^M_{a_{n-k}})$. Finally, let $\mathcal{L}_R = \{L_R(M) \mid M \in \mathcal{M}\}$. Because the labeling of the nodes and of the ports are optimized by $\mathcal{S}$ after choosing $M$, it may happen for two distinct matrices $M, M' \in \mathcal{M}$, that $L_R(M) = L_R(M')$. So, in general $|\mathcal{L}_R| < |\mathcal{M}|$ making harder counting. Let us show that

$$|\mathcal{L}_R| \;\geqslant\; \frac{|\mathcal{M}|}{n!\,(k!)^{n-k-1}}\,. \tag{1}$$

Recall that the routing scheme $\mathcal{S} : \mathcal{F}^M \mapsto (\ell^M, p^M, R^M)$, for every $M \in \mathcal{M}$. Let $\mathcal{S}(\mathcal{U})$ be the range of $\mathcal{S}$. We remark that given an arbitrary triple $(\ell, p, R) \in \mathcal{S}(\mathcal{U})$, there is an unique matrix $M \in \mathcal{M}$ such that $\mathcal{S}(\mathcal{F}^M) = (\ell, p, R)$. To build such a bijection and to find $M$, it suffices to see that for every $i \neq j$, there exists an unique node $b_\alpha \in B$ such that $R(a_i, \ell(a_j)) = p(a_i, b_\alpha)$ (the route from $a_i$ to $a_j$ is forced through a unique node of $B$ by the tree-routing family $\mathcal{F}^M$). Moreover, $M_{i,j} = \alpha$ by construction.

So, $|\mathcal{S}(\mathcal{U})| = |\mathcal{U}|$. Note also that to find $M$, we have restricted our attention to the nodes of $A$ only. The number of node labeling $\ell^M$ for $A$'s nodes is at most $n!/(n - |A|)! = n!/k!$, and the number of port labeling $p^M$ for $A$'s nodes is at most $k!^{|A|}$ (as their degree is bounded by $k$). The number of routing tables $R^M$ for $A$'s nodes is given by $|\mathcal{L}_R|$. It follows that

$$|\mathcal{S}(\mathcal{U})| \leqslant \frac{n!}{k!} \cdot k!^{n-k} \cdot |\mathcal{L}_R|$$

proving Eq. (1) since $|\mathcal{S}(\mathcal{U})| = |\mathcal{U}| = |\mathcal{M}|$.

Let $T_u^M$ be the shortest implementation[9] of $R_u^M$, and let length$(T_u^M)$ denote its length in bits. Let $L_T(M) = (T_{a_1}^M, \ldots, T_{a_{n-k}}^M)$. Finally, let $\mathcal{L}_T = \{L_T(M) \mid M \in \mathcal{M}\}$. Note that there exists a bijection[10] between the sets $\mathcal{L}_T$ and $\mathcal{L}_R$, so $|\mathcal{L}_T| = |\mathcal{L}_R|$. Let us show that

$$\exists M_0 \in \mathcal{M}, \exists u_0 \in A, \quad \text{length}(T_{u_0}^{M_0}) \geqslant \frac{1}{|A|} \log |\mathcal{L}_R| \ . \tag{2}$$

Indeed, assume that there exists some $l$ such that length$(T_u^M) \leqslant l$, for all $M \in \mathcal{M}$ and $u \in A$. Then $\mathcal{L}_T \subseteq (\{0,1\}^l)^{|A|}$. Thus $\log |\mathcal{L}_T| \leqslant l \cdot |A|$, proving Eq. (2). So, combining Eq. (1) and (2), it turns out:

$$\text{length}(T_{u_0}^{M_0}) \geqslant \frac{1}{|A|} (\log |\mathcal{M}| - \log(n!) - (n-k-1)\log(k!)) \ .$$

Noting that $\log |\mathcal{M}| = (n-k)(n-k-1)\log k$, $|A| = n-k$, $\log(n!) \leqslant n \log n$, and that $k < n/2$, we have

$$\begin{aligned} \text{length}(T_{u_0}^{M_0}) &\geqslant (n-k-1)\log k - \left(\frac{n}{n-k}\right) \log n - \left(\frac{n-k-1}{n-k}\right) k \log k \\ &\geqslant (n-2k)\log k - 3\log n \ . \end{aligned}$$

The bound is independent of the routing scheme $\mathcal{S}$. Therefore, we have showed that for every routing scheme, there exists a (shortest path) tree-routing family $\mathcal{F}^{M_0}$ and a node $u_0$ such that the shortest implementation of $R_{u_0}^{M_0}$ requires at least $(n-2k)\log k - 3\log n$ bits. ▮

For unbounded $k$ and $k = o(n)$, the lower bound gives $n \log k - o(n \log k)$, whereas the upper bound we gave in Theorem 1 is $2n \log k + O(n)$. So, our upper bound cannot be improved by a multiplicative factor larger than 2. For planar graphs, we can derive a slightly weaker lower bound choosing $K_{2,n-2}$ (a series-parallel graph): this gives $n - O(\log n)$ bits, whereas our current upper bound is $8n + o(n)$.

Nevertheless, it is still open to find a lower bound that holds routing schemes having the freedom in the choice of the shortest path tree-routing family, or more generally in the choice of the shortest path routing table. To our best knowledge, the only lower bound that holds for *any* shortest path routing tables is the $\Omega(\sqrt{n})$-lower bound of [EGP98] for $n$-node trees.

[9] All implementations we deal with are assumed to be described in a fixed programming language, so can be represented by a binary string of a certain length.

[10] Unfortunately such bijections are uncomputable.

## References

[BHV99] H. Buhrman, J.-H. Hoepman, and P. Vitányi, *Space-efficient routing tables for almost all networks and the incompressibility method*, SIAM Journal on Computing, 28 (1999), pp. 1414–1432.

[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.

[BM99] A. Brodnik and J. I. Munro, *Membership in constant time and almost-minimum space*, SIAM Journal on Computing, 28 (1999), pp. 1627–1640.

[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.

[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, 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.

[FKS84] M. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, Journal of the ACM, (1984), pp. 538–544.

[Gav99] C. Gavoille, *A survey on interval routing*, Theoretical Computer Science, 245 (1999).

[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, 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.

[KK96] E. Kranakis and D. Krizanc, *Boolean routing on Cayley networks*, in 3rd 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 26th 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.

[Mal88] S. M. Malitz, *Genus $g$ graphs have pagenumber $O(\sqrt{g})$*, in 29th 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 28th Annual ACM Symposium on Theory of Computing (STOC), 1996, pp. 392–397.

[MR97] J. I. Munro and V. Raman, *Succinct representation of balanced parentheses, static trees and planar graphs*, in 38th Symposium on Foundations of Computer Science (FOCS), IEEE, ed., October 1997, pp. 118–126.

[Mun96] J. I. Munro, *Tables*, in 16th 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, *Succinct 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 18th Annual ACM Symposium on Theory of Computing (STOC), 1986, pp. 104–108.