# ACM SIGACT News Distributed Computing Column 2

Sergio Rajsbaum*

January 30, 2001

**Abstract**

The Distributed Computing Column covers the theory of systems that are composed of a number of interacting computing elements. These include problems of communication and networking, databases, distributed shared memory, multiprocessor architectures, operating systems, verification, internet, and the web.

This issue consists of two guest contributions: A review of the DISC 2000 Conference by Lisa Higham, and a routing survey by Cyril Gavoille. This survey focuses on routing messages in distributed networks with efficient data structures. After an overview of the various results of the literature, some interestingly open problems are described. Many thanks to Lisa and Cyril!

**Request for Collaborations:** Please send me any suggestions for material I should be including in this column, including news and communications, and suggestions for authors willing to write a guest column or to review an event related to theory of distributed computing. In particular, I would welcome open problems with a short description of context and motivation.

## Report on DISC'00

**Lisa Higham**[1]

The fourteenth International Conference on Distributed Computing, DISC 2000, was held last October 2 through 4 in Toledo Spain.

There were a record number of submissions (over 100) to DISC'00. From these the program committee selected 24 for inclusion as regular papers for the program and proceedings. The conference also featured two invited lectures and seven short papers.

Jean-Claude Bermond presented an invited lecture entitled *Virtual Topologies for Telecommunications Networks*, which surveyed some algorithmic aspects in the design of telecommunication networks. The general problem area can be described as follows. We are given a physical network modelled as a graph $G(V, E)$ with weighted edges, and a collection of requests each modelled as a triple $(s, t, b)$ that represents the need for a logical channel from node $s$ to node $t$ in $G$ with bandwidth $b$. The goal is to satisfy the collection of requests by associating each request $(s, t, b)$ with some path in $G$ from $s$ to $t$ so that no edge carries paths which together exceed its capacity and which minimizes some notion of cost. Typically the problem is further complicated by the need

*Compaq Cambridge Research Laboratory, One Cambridge Center, Cambridge, MA 02142-1612. Sergio.Rajsbaum@compaq.com, rajsbaum@math.unam.mx. On leave from Instituto de Matemáticas, UNAM.

[1]Dept. of Computer Science, University of Calgary, 2500 University Drive N.W., Calgary, Alberta T2N 1N4, CANADA. higham@cpsc.ucalgary.ca

to satisfy additional constraints. Since the problem is NP-hard, we seek heuristics and possibly solutions for very special cases. One technique is to satisfy the constraints on a virtual graph and then embed the virtual graph in the physical one while observing the capacity constraints. Thus step one becomes designing a virtual graph that optimizes various parameters. One example is Virtual Path Layout in Asynchronous Transport mode (ATM) Networks, which aims to embed virtual paths into the network so that the requests can be satisfied by concatenations of these virtual paths while minimizing both the number of concatenations (hop count) for any request and the loads on the physical edges. Other examples concern design and optimization of optical networks that use Wavelength-Division Multiplexing. One problem corresponds to finding a covering of the edges of a logical graph by certain kinds of subgraphs such that each subgraph has a disjoint routing of its edges in the physical network. The goal is to minimize the number of subgraphs in this covering. Another problem requires associating both a path and a wavelength (colour) to each request, typically attempting to minimize the number of colours. Finally, Prof. Bermond observed that in practice it is also very important to consider more realistic functions including, in particular, the cost of the switches.

Sam Toueg presented the second invited lecture. His talk was entitled *Failure Detectors: a Perspective.* PODC papers in 1991 and 1992 by Prof. Toueg and his collaborators established that even weak versions of failure detectors suffice to solve consensus in asynchronous systems with crash failures. Since then his research has explored and made precise a hierarchy of failure detectors and determined what detectors are necessary and sufficient to implement consensus, reliable broadcast and other primitives in asynchronous systems subject to different kinds of failure. We were treated to an overview of all this research. Prof. Toueg described his Heatbeat failure detector, which can be use to implement reliable communication, consensus, k-set agreement, and atomic broadcast in the face of process crashes and message loss. He also introduced a measure of the quality of service for failure detectors in systems with probabilistic message delay and message loss. This in turn admits an algorithm whose parameters can be set to achieve a specified quality of service defined by how quickly the failures are detected and how well false reports of failure are avoided.

The paper *Polynomial and Adaptive Long-Lived* $(2k-1)$*-Renaming* by Attiya and Fouren received the *best student paper award.* Many distributed protocols have complexity that is an increasing function of the number of distinct processes in the system. So to improve efficiency it is appealing to rename the processes that are participating in such a protocol with temporary identifiers from a small name space, while ensuring that all participants hold distinct names. This renaming algorithm should be *waitfree* (no process can hold up any other process) and *long-lived* (processes can dynamically and repeatedly acquire and release names). Furthermore, the cost of renaming should not overwhelm the savings delivered later by the renaming. This implies that the renaming itself should be *adaptive* (have cost that is a function only of the number of processes seeking names) and efficient (polynomial step complexity) and the name space should be as small as possible. All of these requirements are achieved for the first time in this paper. The name space achieve is optimal since any deterministic solution requires at least $2k-1$ names for $k$ participants. Previous waitfree renaming algorithms either used name space size at least quadratic in $k$, or were not long-lived, or had step complexity exponential in $k$. Attiya and Fouren achieve their solution by combining, tuning and enhancing several techniques such as sieves, helpers and timestamps used in previous algorithms. The result is a carefully crafted solution and proof.

The topics addressed by the accepted papers covered a range of topics. Traditional topics such as mutual exclusion, leader election, consensus, and various communication primitives were represented with new results that aimed at more general or realistic models or notions of cost. A few papers studied newer topics related to e-commerce and middleware. Distributed database received some attention. Significant attention went to newer models of distributed computing such as models with new strong atomic primitives, optical models and models with only weak memory consistency. Strong requirements such as self-stabilization and waitfreedom were represented.

Several papers highlight the importance of attention to model details. Merritt and Taubenfeld extend the usual shared memory model to one with no *a priori* bound on the number of processors and permit there to be even an infinite number of participants. In this model they revisit the classical distributed computing problems election, mutual exclusion, consensus and naming to derive a variety of new (im)possibility and complexity results. Buhrman, Panconesi, Silvestri and Vitanyi point out the essential role of distinct identifiers in previous work and show that multi-writers are not always more powerful than single-writers. The assumption that cost of a distributed protocol is dominated by the remote memory accesses, motivates Anderson and Kim to provide a new adaptive read/write atomic mutual exclusion algorithm that confines all busy waiting to spinning on locally accessible variables and minimizes the number of remote memory references. Higham and Kawash determine the number and type of read/write variables necessary and sufficient to solve mutual exclusion when only a weak version of memory consistency, called Processor Consistency is available. Chockler, Friedman, and Vitenberg also look at memory consistency issues. Their paper specifies a small collection of basic consistency conditions, combines them to express several common memory consistency models, and uses them to implement a distributed CORBA caching service. Because sense of direction is another network assumption that can sometimes be exploited to reduce complexity, Boldi and Vigna are motivated to determine the number of edge labels necessary to establish it, and show that even for regular graphs it is much larger than previously conjectured. A team from Sun defended the utility of enhancing the shared memory model with a hardware synchronization primitive called double compare and swap, by showing that it can be used to construct a non-blocking implementation of concurrent double-ended-queues. The model assumed by Malewicz, Russell and Shvartsman assumes that processors may be unable to communicate for prolonged intervals. They characterize the amount of redundant work performed when a collection of processors must jointly complete a collection of tasks in this model.

A few papers made probabilistic assumptions about model properties. Gupta, van Renesse and Birman's paper exploits the practical but seldom used assumption that link and processor failures can be described by some probability distribution to design a probabilistic election algorithm with complexity independent of network size. The paper by Kirousis, Kranakis, Krizanc and Stamatiou contains an algorithm for locating information in a complete network, given that each node's database of where that information is stored is only probably correct.

E-commerce received attention from Blundo, Bonis and Masucci who derived matching communication lower and upper bounds for metering schemes — protocols that securely measure the number of visits received by a web site. Atzmony and Peleg studied the time and communication complexity of bidding in and resigning from distributed auctions. Ungureanu and Minsky discuss the issues and give a case study for business to business electronic commerce where each party is regulated by independent and differing policies.

Instead of using probabilistic gossiping or rumour mongering techniques to broadcast a message over a network, the paper by Lin, Marzullo and Masini suggests a deterministic replacement, showing that it compares favourably in most ways and excels in message complexity. Broadcast is also the topic of the paper of Aguilera, Delporte-Gallet, Fauconnier and Toueg. They formally model a generalization of atomic broadcast (generic broadcast) which ensures that message delivery order adheres only to partial order constraints, define an appropriate measure of the cost of an implementation, and provide thrifty implementations. Another paper dealing with message-passing networks, by Fich and Jakoby, proves that if failures are restricted to only permanent link failures, then end-to-end communication can be achieved in a DAG with single bit headers if each edge has its own message queue whereas $\Theta(\log d)$ bits are necessary and sufficient when nodes have a single input queue and the network has indegree $d$.

Several papers address various versions of fault tolerance. Theel proposed a new strategy, based on techniques from control theory, for designing self-stabilizing protocols. Beauquier, Datta, Gradinariu and Magniette present a transformation for any protocol that stabilizes under a centralized scheduler to one for a distributed scheduler. The paper of Malkhi, Merritt, Reiter and Taubenfeld poses and answers several questions about Byzantine failures in the shared memory setting that were previously explored in message-passing models. Their paper contains several impossibility and possibility results for implementation of shared fault-tolerant objects. The design of fault tolerant optical networks is the topic of the paper by Eilam, Moran and Zaks. Their goal is an approximation algorithm to determine a small set of lightpaths, which when added to an existing set, imparts tolerance of hardware faults to the original collection. Gafni and Lamport's paper extends the Paxos algorithm, which implements any non-Byzantine distributed system by reduction to consensus. In their extension, non-programmable disks can be used for the replication required to achieve fault-tolerance.

Two more papers address maintenance of replicated data in faulty and dynamic settings. If replicated data need only deliver eventual consistency, then it can be maintained with low space and computation overhead, as shown in a new protocol by Saito and Levy that exploits this relaxation of consistency. The algorithms of Patino-Martinez, Jimenez-Peris, Kemme and Alonso maintain traditional consistency, provide fault tolerance, and minimize redundant computation by processing each transaction only once and delivering the result with group communication primitives.

The proceedings of the conference were published by Springer in LNCS number 1914. They were edited by Maurice Herlihy who also chaired the program committee.

Most of us found time outside of the technical sessions to enjoy Toledo. Angel Alvarez was the local arrangements chair. With the help of his team from the Technical University of Madrid, he made our visit to Toledo particularly memorable. It included a evening of banqueting in Spanish style and a fine afternoon on a guided walking tour of old Toledo. The organizers of DISC 2001 have a tough act to follow. But I hear they are up to the challenge. See you in Lisboa in October.

**Routing in Distributed Networks:**
**Overview and Open Problems**

**Cyril Gavoille**[2]

# 1 Statement of the Problem

## 1.1 The routing problem

Delivering messages between pairs of processors is a basic and primary activity of any distributed communication network. This task is performed using a routing scheme, which is a mechanism working in a distributed fashion for routing messages in the network. The routing mechanism can be invoked at any source node and be required to deliver a message to some destination node.

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

Unlikely to the design network problem that is considered usually early in the process of setting up a new network, the problem of designing the management and control systems of the network, including routing, can be designed and optimized after the network construction. The routing problem can be stated as follows: given a graph (the underlying topology of a communication network) fixed in advance, design in each node (i.e., each router of the network) a routing algorithm as efficient as possible. It is required to explicit what we mean by "routing algorithm" and "efficient". A *routing algorithm* is a (computable) function that for each message arriving at a node determines the link on which the message has to be transmitted, and this as function of its destination or any other information contained in the header of the message. The term "efficient" groups a set of desirable quality factors like: the routes generated by the algorithm are (near) shortest paths in the graph; the time to compute the function is low; the number of routes using a same link is low; the size of the data structures required by the algorithm is small; the routing scheme is fault tolerant; and so on.

The way we stated the routing problem is the *static* version: the graph is given in advance and the problem consists to pre-process the graph in order to find some efficient routing schemes on the graph. The dynamic version allows addition and deletion of nodes and/or links in order to model node/link failure and network growing. In this article we will concentrate our attention on the static case. The dynamic case can be tackled by paying more attention on the pre-processing algorithm in charge of the routing algorithm designing. Depending on when failures occur, one can run a distributed pre-processing algorithm to update the routing scheme and to make it adaptive to dynamic networks. It is willing that this maintaining algorithm has low message or time complexity. For more details on the dynamic case, we invite the reader to consult [1, 11], and [2, 5, 28] for end-to-end communication problems, where the goal is to guarantee communications between a fixed pair of nodes in spite of link failures with the minimum memory space in the nodes and minimum communication messages.

To illustrate the (static) routing problem, let us consider the following example: the standard routing algorithm in the Hypercube[3]. Let $x$ denote the binary name of the current node (possibly the source), and let $y$ denote the destination forming also the header of the message currently located in $x$.

ROUTE$(x, y)$: *If $x = y$, then the message is arrived at destination. Otherwise forward it on the edge of dimension $i$ if $x$ and $y$ differ at position $i$.*

If every router possesses a copy of this algorithm, then we obtain a distributed algorithm. The algorithm ROUTE is said a *shortest path* routing algorithm because, one can check that the route generated by the algorithm is the shortest possible one. Let us denote $R_x(y)$ the function that returns, for each destination $y$, the integer $i$ defined by ROUTE$(x, y)$. The function $R_x(\cdot)$ has an argument, $y$, but the algorithm defining $R_x(\cdot)$ depends on $x$ only. In other words the function $R_x(\cdot)$ can be implemented by a program[4] of length $\log n + O(1)$ bits in each node, $n$ being the number of nodes of the Hypercube. Indeed, it suffices to store the name $x$ in the data structure of the program implementing $R_x(\cdot)$ and a constant number of computer basic instructions. Note that such a routing algorithm has therefore a relatively "compact" implementation, and a constant time complexity if basic operations like "=", XOR and integer LOG[5] are available on $O(\log n)$ bit

[3] The nodes of this graph are the $n = 2^k$ binary words of $k$ bits. The node $x_k \dots x_i \dots x_1$ is connected to the node $x_k \dots \overline{x_i} \dots x_1$, for each $i = 1 \dots k$, forming the edge of dimension $i$.

[4] By "program" we mean the set of all data structures and all the control instructions. The log function we consider in this article are in base two.

[5] In order to extract the position of the most significant bit in the result of $x$ XOR $y$.

integers.

## 1.2 Model and terminology

The complexity results are strongly dependent of the routing model (ability to relabel and to assign new addresses to the nodes, size of the addresses, size of the headers, ability to rewrite the headers, etc.). So let us define more precisely all these terms.

Let $G$ be a graph representing a communication network. For the discussion, we assume that graphs are connected, undirected and have $n$ nodes. However, most of the results can be naturally extended to more general model of graphs. Each node $u$ of $G$ has a *name*, an unique identity integer denoted $\mathrm{ID}(u)$. In what follows, we informally confuse between the node $u$ and its name $\mathrm{ID}(u)$. However, the routing mechanism uses a routing-label or *address*, unique for each node and denoted by $\ell(u)$, potentially different of $\mathrm{ID}(u)$.

A *routing function* $R$ on $G$ is a distributing algorithm whose role is to deliver messages between nodes of the network. The algorithm builds a path from the source to the destination, selecting at each intermediate node the next link onto forward the message. Specifically, $R$ consists of a pair of functions $(P, H)$ where $P$ is the *port function* and $H$ is the *header function*. For any two distinct[6] nodes $u$ and $v$, $R$ produces a path or *route* $u = u_0, u_1, \ldots, u_r = v$, a sequence $h_0, h_1, \ldots, h_r$ of headers, and a sequence $p_0, p_1, \ldots, p_r$ of output port numbers. The length of the route, denoted $\rho_R(u,v)$, is the cost of the path from $u$ to $v$ if $G$ is weighted, and otherwise $\rho_R(u,v) = r$. The port numbers identify the links connected to a node. It is a local name, so that a link connecting $x$ to $y$ may have a different name in $x$ (output port) and in $y$ (input port). The port numbers are unique integers taken from the set $\{1, \ldots, d\}$, where $d$ is the number of ports corresponding to the degree of the current node[7]. A message with header $h_i$ arriving at node $u_i$ through input port $q_i$ is given a new header $h_{i+1} = H(u_i, q_i, h_i)$, and is forwarded on the output port $p_i = P(u_i, q_i, h_i)$. Thus, we require that for every $i \in \{0, \ldots, r-1\}$, $H(u_i, q_i, h_i) = h_{i+1}$, $P(u_i, q_i, h_i) = p_i$ 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 $u_{i+1}$ (see Fig. 1). On each router, there exists a special link numbered 0. It insures the communication between the router and its host associated at the node $u_i$. This allows us to complete the description by imposing the constraints that $q_0 = p_r = 0$, as well as $h_0 = \ell(v)$, thus fixing the initial header, which is provided to the router from its host (see Fig. 1).

This mathematical formulation allows us to compare in a precise way all the results (specially the lower bound results) and all the routing strategies of the literature.

A *routing strategy* is an algorithm that computes for a graph $G$ a routing function $R$ on $G$. Hence the strategy consists of a pre-processing during the set-up time of the graph and is responsible of the addresses assignment, port labeling, and distributed data structures construction required by the routing scheme. (We use the term *routing scheme* to deal with an implementation of a routing function.) A routing scheme, and more generally, a routing strategy is *name-independent* if "names" does not change in the pre-processing, that is, the address $\ell(u)$ is simply its original name $\mathrm{ID}(u)$. A strategy is *universal* if it provides a routing scheme for any graph. We denote $R_u$ the restriction of $R$ to $u$, so-called *local routing function*.

A routing function that, in each node, does depend of the header only, and not of the input port, is said *oblivious*. And, an oblivious function that depends of the destination only, i.e., such

[6]Traditionally, we never consider the routing from $u$ to it-self, this because it is assumed that the host processor connected to the router has enough information and computational power to avoid this kind of useless communications. Moreover, the case $v = u$ makes troubles for the definition of the *stretch factor* for which the distance from $u$ to $v$ is taken as the denominator of a fraction.

[7]In directed graphs, we have to consider the in- and out-degree.

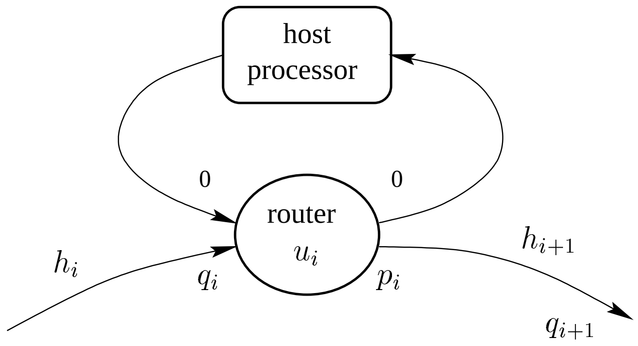


Figure 1: A general model of router.

that $h_0 = h_1 = \cdots = h_r = \ell(v)$, is a *direct routing scheme*. Finally, a direct scheme that uses the address range $[1, n]$ is called a *routing table*.

All theses refinements and considerations have an impact on the implementation of the routing schemes. The ability of header modification for instance, can be costly for optical networks that would require electronic-optic conversions. Direct schemes are the simplest ones and have also the *loop-less* property: the messages following the route can never cycles, since otherwise they would loop forever contradicting the routing function definition (there must exist a path between any pair of nodes).

## 1.3 Complexity measures

As we will see there are relationship between the length of the routes generated (near or far from the shortest paths) and the size of the local data structures used by a routing scheme (i.e., the available knowledge). Actually, the trade-off between the computational power and the size of the knowledge is a central theme in Theory of Distributed Computing. Routing with the most compact distributed data structures is a perfect illustration of this paradigm[8].

Let $R$ be a routing scheme on a graph $G$. The *stretch factor* of $R$ is the value defined by $\max_{u \neq v} \rho_R(u, v)/d_G(u, v)$, where $d_G(u, v)$ denotes the length of a shortest path from $u$ to $v$ in $G$ (the cost of a minimum path in the weighted case). A routing scheme of stretch factor 1 is termed a *shortest path* routing scheme.

The *memory space* of $R$ is the size (in bits) of all the data structures it uses. One can distinguish the *local* memory space of $R$ in a node $u$, and the *total* memory space of $R$, defined as the sum of the local memory space of all the nodes of $G$. As we will see, the memory space may depend of the size of the headers and the size of the addresses. Note that, a priori, it is not required that the size of local memory space of $u$ is at least as large than the size of the address of $u$.

The *routing time* (sometimes called *latency*) is the maximum of the worst-case time complexity of $R_u$, the maximum taken over all node $u$ of $G$. The *total routing time* is the maximum of the sum of the time complexity for the local routing decisions performed along a route, the maximum taken over all the routes.

[8]The "Best Student Paper Awards" of the 1996 and 2000 editions of the annal *Symposium on Principles of Distributed Computing (PODC)* deal with compact routing, cf. [25, 26].

# 2 Overview

## 2.1 Universal routing schemes

First, remark that every graph has a shortest path routing scheme with (local) memory space of size $O(n \log d)$ bits for each node of degree $d$: it suffices to use routing tables. (One can list in each source the right output port for the $n-1$ possible destinations. The direction of each destination can be determined by rooting a minimum spanning tree in the source.) Thus, the use of routing tables is an universal routing strategy.

After this remark, one may naturally ask whether there exists universal routing strategy that are more compact? Say more compact than $O(n \log d)$ bit per node.

There are at least two ways of designing shortest path and compact routing tables: find a suitable address set and a suitable system of shortest paths for each source. The idea behind naming nodes with a suitable address is to encode useful information about the network and then to make use of this implicit information when performing the routing. Clearly, a routing strategy that does not allow renaming of nodes of a ring cannot avoid a $\Omega(n)$ bit lower bound for the memory space. (If the nodes are permuted at random, a source $x$ needs to store $\Omega(1)$ bit for each destination $y$ to determine whether a message has to be forwarded to its left or to its right.) Note that the original node name can always be kept in the final address of the router, for instance setting $\text{address}(u) = \langle \text{ID}(u), \ell(u) \rangle$ where $\ell(u)$ is the routing-label. That is why we pay more attention on $\ell(u)$, and try to minimize its size. Shortest path selection by the routing strategy is desirable as well. A routing strategy that does not give this ability provides a $\Omega(n)$ bit lower bound of memory space for a $K_{2,n-2}$ (a complete bipartite graph). (Consider two nodes $x$,$y$ of the largest part. If the shortest path from $x$ to $y$ is fixed by a coin flip (there are two shortest paths from $x$ to $y$ using distinct first edges) and is not optimized by he routing strategy, $x$ would require to store $\Omega(1)$ bits for $y$.) Obviously, rings and complete bipartite graphs support shortest path routing schemes with $O(\log n)$ bits of memory space if renaming and shortest path selection is allowed.

These two kinds of optimizations makes interesting the problem, in particular for specific families of graphs like trees [34, 35], outer-planar and bounded genus graphs [17, 23], $k$-trees [31], etc. However, most of the routing strategies proposed are rather specific, and thus not universal. Unfortunately, in [25], we negatively answer to the question of universal and compact routing strategies. We showed that for every integer $d$, $3 \leqslant d \leqslant n/2$, every shortest path universal routing strategy requires $\Omega(n^2 \log d)$ bits of total memory space for some worst-case graph of maximum degree $d$, assuming that addresses can be optimized by the strategy and are taken in the set $\{1, \ldots, n\}$. (Note that port numbers and shortest paths are also selected and optimized by the routing strategy.) Therefore, this shows, up to a constant multiplicative factor, the incompressibility of shortest path routing tables. The result can be extended to any routing strategy generating optimal addresses up to $c \log n$ bits, for every constant $c \geqslant 1$, and whatever is the header size.

It turns out that memory space can be reduced only if we accept to relax at least two constraints: the shortest paths and the size of the addresses.

## 2.2 Memory space *vs.* stretch factor

The issue of saving space in routing schemes by settling near-shortest routes was first raised in [27]. The proof of a trade-off between the memory space and the stretch factor has been given in [32, 33]. For every $k$, it is shown that every universal routing strategy of stretch factor $\Theta(k)$ requires a total memory space of $\Omega(n^{1+1/\Theta(k)})$. This result is completed by a universal routing strategy of stretch factor $\Theta(k)$ with total memory space $O(n^{1+1/\Theta(k)})$. Though this strategy is almost optimal in terms of its efficiency-space trade-off, it has few drawbacks.

Looking in more details the hidden constants with the $\Theta$ notation, we can observe that the bounds are not tight. More precisely, the memory space lower bound is $\Omega(n^{1+1/(2s+4)})$ for arbitrary stretch $s \geqslant 1$. The upper bound results of hierarchical cluster decomposition providing an effective stretch factor $s = 12k + 3$ for some integer parameter $k \geqslant 1$, for a total memory space of $O(k^3 n^{1+1/k} \log n)$, for addresses of size $O(\log^2 n)$ bits, and for headers of size[9] $O(\log n)$ bits. For instance, if we would like to design routing schemes with total memory space smaller than routing table one, i.e., smaller than $O(n^2 \log n)$ bits, we must choose $k = 2$ in order to obtain $O(n^{1.5} \log n)$ bits of space but we pay in this case a stretch $s = 27$. On the other hand, the memory space lower bound for $s = 27$ is $\Omega(n^{1+1/58})$ only. This motivated several works for improving the trade-off bounds.

At the present time, the best lower bounds indicate that for every $s < 1.4$ the total memory space has to be $\Omega(n^2 \log n)$ [25], and, for $s < 3$, the total memory space has to be $\Omega(n^2)$ [22].

The strategy proposed in [32] is not name-independent, assume unit cost on the link of the network, and does not bound the local memory space (the local space can be as larger than $O(n \log n)$ bits for some nodes).

Other methods that overcome these problems achieve an inferior efficiency-space tradeoff. The hierarchical routing strategy presented in [4] uses $O(k\, n^{1/k} \log n)$ bits of local memory and guarantees a stretch factor $O(k^2\, 9^k)$. The routing strategy presented in [6] retains the advantages of the former one, while regaining the polynomial trade-off. In particular it guarantees, for every integer $k \geqslant 1$, a stretch factor of $O(k^2)$, while using $O(k\, n^{1/k} \log^2 n \log D)$ bits of local memory, where $D$ is the weighted diameter of the network.

The major disadvantage of all the proposed hierarchical routing strategies is that they are rather complex, and thus they might be impractical especially for high-speed networks for which the routing time in each node must be very short. Broadly speaking, the hierarchical routing strategies are based on techniques for generating a sparse cover of clusters for the underlying graph. In order to implement the routing scheme, there are $k = O(\log n)$ levels of covers with increasing radii. Intuitively, higher levels are responsible for routing to farther destinations. A message is first sent on the lowest level, on the hope that the destination is nearby. If this is not the case, then the transmission might fail, in which case the message will bounce back to the originator. This step may be viewed as one "phase" of the routing process. Once receiving the message back (with a notification of failure), the originator tries to send it on a higher level, and so on, until the routing process succeeds in delivering the message. Clearly, the implementation of this procedure in every router results in a complex decision function; the number of the present phase has to be coded into the header of the message, thus a new message header must be recomputed and rewritten by the originator upon each retransmission, i.e., in every phase of the algorithm. Moreover, intermediate routers must also change the message header, for example, in order to notify that a failure occurred. (Another possibility is to send only a failure notification, but in this case the message originator must keep a copy, and an intermediate router has to generate additional messages). In addition, those strategies treat the nodes of the network non-uniformly, in the sense that different nodes play different roles, thus the decision function could be substantially different at different nodes. As high-speed networks gain popularity and increase in size, these drawbacks become crucial, since the main routing bottleneck in these networks is often the decision function in the nodes and not the propagation delay. Therefore, simple routing schemes, like direct schemes which could be

[9] The reader should not be surprised by headers shorter than addresses. The source node, upon reception of the destination address, say $h_0$, coming from the host on input port 0, has the opportunity to modify $h_0$, and to create the first header $h_1$ of shorter size.

implemented in hardware may be preferable in practice.

### 2.3 Direct routing schemes and low stretch factor

Subsequently to the previous discussion, considerable attention is given recently to an opposing design philosophy, focusing on simple and direct schemes. These schemes employ a simple "transmit and forget" type decision function in the nodes, depending only on the destination of the message, and the destination is the only information coded in the message header (which is determined once and for all by the originating router, and is never changed afterwards.) They are loop-free and can be implemented by some routing tables (that are sometimes compacted, but with a relatively low routing time). That is why, other routing strategies have been designed, in particular some routing strategies with small stretch factor $s \in [2,5]$.

In this framework, [10] proposed direct loop-free routing schemes for weighted graphs with $O(n^{2/3}\log^{4/3} n)$ local memory space. The stretch is at most $s = 3$, and addresses and headers are of size $3\log n$. The space bound can be reduced to $O(\sqrt{n}\log^{3/2} n)$ bits if one accept a small increasing on the stretch to $s = 5$ [12]. The latter routing strategy is based on routing tables (the tables are compacted into intervals of integers, namely these are *interval routing schemes*, cf. [21] for a survey of this technique). Thus they are loop-free, and use headers/addresses which are taken from the set $\{1,\ldots,n\}$, i.e., on $\log n$ bits exactly. It is also remarked that the stretch is, in average on all the source-destination pairs of the graph, bounded by $\bar{s} = 3$. Moreover, the longest route does not exceed $2D$ ($D$ being the weighted diameter of the graph), and is even bounded by $\lceil 1.5D \rceil$ in the case of uniform weights. The routing time is $O(\log n)$.

Actually, the bound on the local memory space is almost optimal. It is shown in [12] that no loop-free routing strategy with address range $[1,n]$ can guarantee a local memory space lower than $c\sqrt{n}$ bits[10] on every family of graphs including trees. The result holds for every stretch factor, since on trees a loop-free routing scheme of stretch $s$ is a routing scheme of stretch 1. It follows that the trade-offs presented previously (for instance the name-independent hierarchical routing strategy of [6]) cannot pretend to loop-free routing schemes, thus require to change and rewrite the headers at least once.

**Open question**

- What is the best trade-off between local memory space and stretch factor (or average stretch factor) for universal direct routing schemes using addresses taken in $\{1,\ldots,m\}$, with $m > n$?

The same question arise for universal $k$-phase routing strategies, namely strategies that provide routing schemes for which the header of each message can be rewritten at most $k$ times along its route.

### 2.4 Almost all the graphs

We just have seen that in the worst-case a shortest path routing scheme requires $\Theta(n\log n)$ bits of local memory space (cf. the result of [25] taking $d = \Theta(n)$). But, are such worst-case networks rare? Is the situation better for the "average case"? The answer is yes. Actually, surprisingly enough, the networks that require a large memory space for routing along shortest paths are not the ones that possess the maximal entropy[11]. Graph structures that make difficult the routing are

[10] Precisely, $c = (\pi\sqrt{2/3})/\ln 2 = 3.7006565593\ldots$

[11] This family of graphs is quite hazy, but it can be viewed as the set of graphs whose adjacency matrix is not compressible in the Kolmogorov Complexity sense.

not completely random. As we will see, on the contrary, a graph with a fully random structure has a shortest path routing scheme with $O(n)$ bits of local memory space.

Certain results in graph theory are valid for "almost all the graphs". The term "almost all" is statistical. It means that the fraction of $n$-node graphs for which the property holds tends to 1 as $n$ tends to infinity. The tools to establish such kind of results are probabilities, with the family $\mathcal{G}_{n,p}$ of random graphs[12], or the Kolmogorov Complexity [29] with the Kolmogorov random graphs. These two tools are very close in essence.

In [13] the ability of random graphs in $\mathcal{G}_{n,p}$, for some particular values of $p$, to support shortest path routing tables that can be compacted into intervals has been considered. More generally, and using Kolmogorov random graphs, [9] showed that a fraction of at least $1 - 1/n^3$ of all the graphs has a shortest path routing table of size $3n + o(n)$ bits (per node) under the assumption that node address range is $[1, n]$ and node addresses are randomly permuted, and that each node knows its neighborhood for free. However, if the addresses are on $c \log^2 n$ bits, where $c$ is a constant, then the routing table can be reduced to $c \log^2 n$ bits only. Other results are mentioned for stretch factor $s > 1$. Finally, in [24] the $3n + o(n)$ bit upper bound has been slightly improved: for a fraction of at least $1 - 1/n$ of all the graphs support shortest path routing tables of size $n + O(\log^4 n)$ bits for addresses taken in the range $\{1, \ldots, n\}$.

**Open question**

- Is the $n + o(n)$ upper bound is the best possible one for a fraction of $1 - o(1)$ of all the graphs?

**Remark.** To design such a lower bound on the memory space is harder than it looks. First, a node does not need to know its neighborhood (even if the neighborhood results of a random choice of $n/2$ nodes among $n - 1$). We can relabel the ports according to some information of the neighbors and it may decrease the information even bellow the degree of the node (for instance, it is shown in [21] that trees have a $O(\sqrt{n})$ bit local memory space routing scheme even for large degree node). Secondly, in a random graph, node relabeling changes the probability to have a connection between two arbitrary nodes labeled respectively $x$ and $y$ in $\{1, \ldots, n\}$.

## 2.5 Separator, planar and bounded genus graphs

There are strategies that are not universal, but very efficient for specific class of graphs.

In [19] it is presented routing schemes for the family of graphs that are recursively decomposable by a separator of size at most $c$. A separator is a subset of nodes whose removal disconnect a graph in two (or more) connected components, each one of size at most $2/3$ of the initial size of the graph. More generally, the graph is said $c$-decomposable if for every node weight assignment of the graph there exists a separator of size $c$ that provides connected components of weight (the sum of the weight of the nodes in the component) at most $2/3$ of the weight of the whole graph. This definition implies a recursive decomposition of the graph with at most $\log_{3/2}(n)$ hierarchical levels, cf. [19]. A separator insures that every route between two nodes of distinct components has to cross some nodes of the separator. If the separator is small in size, this allows to concentrate the routing information towards the nodes of the separator. Outer-planar graphs, and more generally, series-parallel graphs are 2-decomposable, graphs of treewidth bounded by $k$ are $O(k)$-decomposable, planar graphs are $O(\sqrt{n})$-decomposable, and more generally, graphs of genus bounded by $g$ are $O(\sqrt{gn})$-decomposable.

[12] In this model, graphs have $n$ nodes and with probability $p$ there is an edge connecting two nodes of the graph, cf. [7].

More precisely, [19] proposed two routing strategies. The first one, applicable to edge-weighted graphs, uses a total memory space of $O(cn\log^2 n)$, a stretch factor $s=3$, and addresses of size $r\log n$ bits, $r>1$ being a small constant. The second one, with the same memory space, decreases the stretch to $s=1+2/\alpha$, where $1<\alpha\leqslant 2$ (thus $s<3$) is the root of $\alpha^{\lceil(c+1)/2\rceil}-\alpha=2$, for an increasing of the addresses size to $3.42\,c\log c\,\log n$ bits. For $c\in\{2,3\}$, $\alpha=2$, and for $c\in\{4,5\}$, $\alpha\leqslant 2.32$. The routes are not necessary loop-free, the headers are supposed to be rewritable, and the local memory space is not bounded.

These strategies are efficient only if $c$ is constant. For planar graphs, $c=\Theta(\sqrt{n})$, the same authors proposed in [18] two better strategies. Still for weighted graphs, the first one has $O(n^{4/3}\log n)$ total memory space and a stretch factor of $s=3$, for addresses and headers of size $O(\log n)$. The second one, for every constant $\epsilon$, $0<\epsilon<1/3$, provides a total memory space of $O((1/\epsilon)n^{1+\epsilon}\log n)$ bits for addresses of size $O((1/\epsilon)\log n)$, and a stretch factor $s=7$. Both strategies suffer of the previous drawbacks: they do not bound the local memory space (that can be as larger than $O(n\log n)$), and use rewritable headers and addresses of size strictly longer than $\log n$. All the strategies presented in [18] and in [19], have routing time linear in the size of addresses, that is $O((1/\epsilon)\log n)$.

The case of shortest path routing schemes for planar graphs ($s=1$) has been studied in [23]. It is proposed a direct routing scheme (a routing table with addresses and headers are taken in the range $[1,n]$) with local memory space $8n+o(n)$ bits and with $O(\log^{1+\epsilon} n)$ routing time, for every constant $\epsilon>0$. The scheme applies to weighted graphs, and to any given tree-routing family, i.e., a family of $n$ spanning trees, each tree being rooted in a unique node of the graph and defining the routes to all the destinations. In this model the strategy cannot optimize the shortest path selection, and thus the $\Omega(n)$ lower bound of the complete bipartite $K_{2,n-2}$ occurs here (cf. Paragraph 2.1). The result extends naturally to graphs of genus at most $g$, increasing the local memory space to $O(n\log g)$ bits. All the schemes are extended, with the same performances to graphs having at most $o(n\log g/\log n)$ edge crossing. It is interesting to note that the results of [23] do not use combinatorial separability of bounded genus graphs, but are obtained by the use of $k$-page embedding, a geometric representation of graphs.

**Open questions**

• What is the local memory space complexity of shortest path routing tables in planar graphs? (The current lower bound is only $\Omega(\sqrt{n})$ bits [12] for strategies optimizing shortest paths, and the upper bound is $O(n)$ [23]).

• More generally, what is the complexity of the local memory space for shortest path routing tables in graphs of genus bounded by $g$? (Indeed, it is not clear that $O(n\log g)$ is the best possible upper bound. A smaller dependency in $g$ is possible, no lower bound greater than $\Omega(\sqrt{n})$ exists).

# 3 Some Key Problems

In this section we stress several problems for routing in distributed networks.

## 3.1 How to find the right interval?

Assume that a graph has a routing table $R$ such that, in each source, the set of addresses destinations using the same output port consists of a single interval of consecutive integers, i.e., $R$ can be implemented by an interval routing scheme. The question is to construct an efficient data structure for this scheme. From practical point of view, the answer is important.

Let us consider a node $x$ of degree in this graph, and let $[a_i, b_i]$ denote the interval assigned to the arc $(x, y_i)$, $i \in \{1, \dots, d\}$, ordered such that $a_1 < a_2 < \cdots < a_d$. To answer to the routing query, we need to compute as quick as possible, the index $i$ such that $y \in [a_i, b_i]$, for every $y \in \{1, \dots, n\}$. Let us denote $R_x(y) = i$ this function.

A first solution simply consists in storing the $a_i$'s in a table $T$ such that $T[i] = a_i$. The computation $R_x(y)$ consists in a binary search of $y$ such that $a_i \leqslant y < a_{i+1}$, because the $a_i$'s form a partition of the range $[1, n]$. The routing time of $R_x$ is $O(\log d)$, and its memory space is $O(d \log n)$ bits. However, if $d > n/\log n$, one can do smaller and faster.

The second solution guarantees a memory space of $n + o(n)$ bits with a $O(1)$ routing time. It suffices to represent the set of $a_i$'s by a binary string, $B$, such that $B[\alpha] = 1$ if $\alpha \in \{a_1, \dots, a_d\}$ and $B[\alpha] = 0$ otherwise. It turns out that $R_x(y) = i$ if the number of $a_i$'s less or equal to $y$ is exactly $i$. It corresponds also to the number of 1's in $B$ up to position $y$. The memory space is $n$ bits (the string $B$) and the routing time is a priori $O(y)$, the traversal time of $B$ up to $y$. In [30] it is shown that this type of queries can be solved in constant time[13] in the worst-case, thanks to a data structure of size $n + o(n)$ bits (moreover constructible in polynomial time).

Obviously, the ideal solution would be a compact representation of the set $\{a_1, \dots, a_d\}$ by a data structure of size[14] $\log \binom{n}{d} = \Theta(d \log(n/d))$ allowing constant routing time. In the same spirit, [8] proposed a quasi-optimal coding of integer sets (up to a multiplicative constant) with constant time for membership queries. The question of computing the rank of an element is open.

**Open question**

- Is it possible to find a data structure of size at most $O(d \log(n/d))$ bits per node of degree $d$, and a constant routing time for graphs supporting an interval routing scheme?

### 3.2 Total routing time

We saw that it is not easy to design a compact data structure for a minimal routing time, even for the case of interval routing scheme. An alternative would be to consider the total routing time on a route of length $L$.

Consider the "standard" shortest path routing in the de Bruijn graph. The nodes of this graph are the $n = 2^k$ binary words of length $k$. The node $x_k x_{k-1} \dots x_2 x_1$ is connected to the nodes $x_{k-1} \dots x_2 x_1 \alpha$, for $\alpha \in \{0, 1\}$. It consists to compute the largest prefix of the destination address that is a suffix of the source address (addresses correspond to node names). This prefix constitutes the first header. At each intermediate node, the first bit of the current header is extracted: if the bit is 0 the message is forwarded to output port 1, if the bit is 1 it is forwarded to port 2 (this graph is directed and has only two outgoing arcs). In both cases, the extracted bit is destroyed and the new header is one bit less. The message arrives at destination when the header is empty. The total routing time on a route of length $L$ is $O(L + \log n)$, the $\log n$ term coming from the computation of the first header that can be performed by the Boyer-Moore's algorithm [3]. Note that the routing time is constant excepted in the source. Since $L \leqslant \log n$ (the diameter is $k$), the total routing time never exceeds $O(\log n)$.

[13] The computation model is the word-RAM model in which standard arithmetic and bitwise logic operations on integers of $O(\log n)$ bits run in a unit of time.

[14] This is an optimal coding since there are $\binom{n}{d}$ possible sets.

The problem to design a compact data structure in order to optimized the total routing time has been first pointed in [16]. For weighted outer-planar graphs (that all support a shortest path interval routing), he presents a data structure of size $O(d \log n)$, mainly based on intervals with auxiliary tables in a way that the total routing time never exceeds $O(L+\log n)$ for messages between nodes at distance $L$.

Expressed in an other complexity measure, the bit-operations model, [14] showed that every graph of diameter $D$ support a routing scheme with routing time $O(\log n)$ bit-operations and with total routing time $O(D + n^{1/k} \log n)$ bit-operations, where $k \geqslant 2$ is an arbitrary constant. Note that in this complexity measure, standard routing tables have total routing time $\Theta(D \log n)$ bit-operations. Indeed, reading of $\ell$ bits in a table of size $t$ costs $O(\ell + \log t)$ time, thus it costs $\Theta(\log n)$ bit-operation to read the output port in a standard routing table[15]. The technique to save time is therefore to take few routing decisions (at most $O(n^{1/k})$ for some parameter $k$). Most of the routers takes their own decision from few bits of the header only. Note that in the result [14] the routes are not shortest paths, but their length are bounded by $O(D)$.

**Open question**

• Is there any universal routing strategy with total routing time bounded by $O(D + \log n)$ bit-operations?

### 3.3 Routing in trees

Routing in trees is a basic and important problem. Indeed, most of the hierarchical routing strategies are based on tree covering. Often, at the final phase of the routing protocol, the problem to route in a small region spanned by a tree occurs. It is quite easy to design space-efficient routing scheme for trees. For instance, interval-based routing (assuming addresses fixed by a DFS traversal numbering, each sub-tree defines an interval of consecutive addresses) achieves a total memory space of $O(n \log n)$, thus $O(\log n)$ bits per node in average. This scheme is easy to implement and the routing time is $O(\log n)$. however one can object the two following remarks: (1) the local memory space is not bounded by $O(\log n)$ when the degree is large; (2) one could expect a constant routing time with a better data structure.

In Paragraph 3.1, we saw that in the case of interval routing scheme, one can get a local memory space $O(d \log(n/d))$. With a space $n + o(n)$ one can even guarantee a constant routing time. Is the space $O(d \log(n/d))$ bound optimal for trees?

The answer is no: the right bound is $O(\sqrt{n})$. In [21] it is shown that, thanks to a DFS numbering according to the number of descendents of each sub-tree, the local routing function in $x$ can be computed from a sequences $S_x = (n_1, \ldots, n_d)$, with $n_1 \leqslant \cdots \leqslant n_d$. Here, $n_i$ is precisely the number of descendents in the sub-tree rooted at the $i$th child of $x$. Because $1 \leqslant n_1 \leqslant \cdots \leqslant n_d$ and $\sum_{i=1}^{d} n_i = n - 1$, the sequence $S_x$ can be coded with at most $O(\sqrt{n})$ bits, since the number of such sequences (called partitions of $n-1$) is bounded by $2^{\Theta(\sqrt{n})}$. Thus the local memory space of $x$ is bounded by $O(\sqrt{n})$, and this is actually the optimal bound, cf. [12]. Roughly speaking, the routing function in $x$ for a destination of address $y$ consists to compute the index $i$ such that $y - x \in (\sum_{j=1}^{i-1} n_j, \sum_{j=1}^{i} n_j]$.

[15] In this model, reading or writing a single bit of header costs one unit of time. However, the message and the bits of the header that are not read can be copied and transmitted onto the outgoing link without any penalties – in the de Bruijn example, the routing time is $O(1)$ bit-operations though headers are non constant

**Open question**

• Design a compact data structure for partitions of $n$ using optimal space, and allowing constant time rank query type?

### 3.4 Exponential routing time and Cayley graphs

In an extension of [14], one can show that there are graphs having shortest path routing tables of size $O(\log n)$ for each node with the following property: every shortest path routing scheme using less than $cn \log n$ bits of local memory space, for a suitable constant $c > 0$, must have a routing time greater than any constant size stack of exponentials, i.e.,

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

It is clear that if the data structure is to compact, the time to extract some piece of information can be very large.

Cayley graphs are precisely a family of graphs supporting a theoretical low local memory space. They have strong regularity property (based on a group structure), and thus are good candidates for compact routing tables since they their adjacency matrix can be entirely described with a few number of bits. However, from such global information (say, the matrix of the graph), there is no efficient way to extract a shortest path, or simply the first edge of a shortest path. Routing schemes need local information.

More precisely, nodes of a Cayley graph are element of a group $\Gamma$ and the arcs[16] are defined by a given set of generators $S \subset \Gamma$: $x$ is connected to $y$ if there exists an element $s \in S$ such that $y = x + s$.

Consider the following example: $\Gamma = \mathbb{Z}$ (the additive group modulus $n$) and $S = \{\pm 1, \pm c_1, \pm c_2\}$ with $c_1, c_2 \in \mathbb{Z} \setminus \{0, 1\}$ (we make the graph symmetric taking opposed generators). The Cayley graph $(\Gamma, S)$ can be described by given the two integers $c_1$ and $c_2$. Hence, the shortest path routing can be solved with $O(\log n)$ bits of memory space. Indeed, one can rebuild the whole graph (in the router memory), and apply a standard shortest path algorithm in order to extract the first edge of a shortest path. The point is that this method would require $\Omega(n)$ routing time, whereas one can expect a poly-logarithmic routing time, the size of the input being $O(\log n)$. Unfortunately, one need to solve the minimal decomposition of an element in sum of generators, a difficult problem. In fact, for $S = \{\pm c_1, \pm c_2\}$ the problem can be solved in $\log^{O(1)} n$ time, but is still open for $S$ of larger cardinality.

**Open question**

• What is the best routing time we can achieve for shortest path routing scheme on Cayley graphs of degree $k$ and defined on Abelian groups, if the local memory space if bounded by $O(k \log n)$.

**Remark.** The allowed spaced is enough to store all the generators, and whole the graph information: there are $O(k)$ generators in $S$, each one can be described by an integer taken in $[1, n]$, and one can show that there are at most $n^{O(1)}$ non-isomorphic Abelian groups with $n$ elements.

[16] Cayley graphs are directed. However if $-s \in S$ for every $s \in S$, one can considered them as undirected graphs.

### 3.5 Routing and other distributed tasks

It is worth to observe that design a compact data structure for routing in a distributed network is a difficult task. For instance, to determine the minimum number of intervals for which the graph has a shortest path $k$-interval routing scheme is NP-hard (with at most $k$ intervals per link). The pre-processing on the graph to optimize routing is time consuming in general. A natural question is thus to ask if such efficient data structures, once generated by the pre-processing algorithm, could be useful to other distributed tasks than routing, e.g., broadcasting or leader-election? or if with a little effort one could not modify the compact data structures allowing fast multiple queries in addition to routing.

In [15], we positively answer to this question for the case of shortest path 1-interval routing schemes. Mainly, it is shown that there are simple broadcast algorithms that allows broadcasting message from any source in at most a total of $O(n)$ messages, and using the routing information only. It implies also $O(n)$ message algorithm for leader-election improving the first contribution in this routing and election problem, [35]. Note that with no specific information, the leader-election problem has already $\Omega(n \log n)$ message-complexity lower bound for a ring [20].

#### Open question

• Does the message-complexity remain in $O(n)$ for $s$-stretched $k$-interval routing schemes for constant $k$ and constant $s$?

The problem is open for other representation of compact routing schemes. For instance it is not clear if a graph having low local memory space for routing, say $O(\log n)$ bits, has also some ability to broadcast or elect a leader with low message-complexity.

## References


[1] Y. Afek, E. Gafni, and M. Ricklin, *Upper and lower bounds for routing schemes in dynamic networks*, in $30^{th}$ Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Oct. 1989, pp. 370–375.

[2] Y. Afek, E. Gafni, and A. Rosén, *The slide mechanism with applications in dynamic networks*, in $11^{th}$ Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, Aug. 1992, pp. 35–46.

[3] A. V. Aho, *Algorithms for finding patterns in strings*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. A, North-Holland, 1990, pp. 255–300.

[4] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, *Compact distributed data structures for adaptive routing*, in $21^{st}$ Symposium on Theory of Computing (STOC), vol. 2, May 1989, pp. 230–240.

[5] B. Awerbuch, Y. Mansour, and N. Shavit, *Polynomial end-to-end communication*, in $30^{th}$ Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Oct. 1989, pp. 358–363.

[6] B. Awerbuch and D. Peleg, *Sparse partitions*, in $31^{th}$ Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, 1990, pp. 503–513.

[7] B. Bollobás, *Random Graphs*, Academic Press, New York, 1975.

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

[9] 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.

[10] L. J. Cowen, *Compact routing with minimum stretch*, in $10^{th}$ Symposium on Discrete Algorithms (SODA), ACM-SIAM, 1999, pp. 255–260.

[11] S. Dolev, E. Kranakis, D. Krizanc, and D. Peleg, *Bubbles: Adaptive routing scheme for high-speed dynamic networks*, in $27^{th}$ Annual ACM Symposium on Theory of Computing (STOC), May 1995, pp. 528–537.

[12] 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, Aug. 1998, pp. 11–20.

[13] M. Flammini, J. van Leeuwen, and A. Marchetti-Spaccamela, *The complexity of interval routing on random graphs*, in $20^{th}$ International Symposium on Mathematical Foundations of Computer Sciences (MFCS), J. Wiederman and P. Hájek, eds., vol. 969 of Lecture Notes in Computer Science, Springer-Verlag, Aug. 1995, pp. 37–49.

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

[15] P. Fraigniaud, C. Gavoille, and B. Mans, *Interval routing schemes allow broadcasting with linear message-complexity*, in $19^{th}$ Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, July 2000, pp. 11–20.

[16] G. N. Frederickson, *Searching among intervals and compact routing tables*, Algorithmica, 15 (1996), pp. 448–466.

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

[18] ——, *Efficient message routing in planar networks*, SIAM Journal on Computing, 18 (1989), pp. 843–857.

[19] ——, *Space-efficient message routing in c-decomposable networks*, SIAM Journal on Computing, 19 (1990), pp. 164–181.

[20] G. Gallager, P. A. Humblet, and P. M. Spira, *A distributed algorithm for minimal spanning tree*, ACM Trans. Programming Languages Systems, 30 (1983), pp. 66–77.

[21] C. Gavoille, *A survey on interval routing*, Theoretical Computer Science, 245 (2000), pp. 217–253.

[22] C. Gavoille and M. Gengler, *Space-efficiency of routing schemes of stretch factor three*, in $4^{th}$ International Colloquium on Structural Information & Communication Complexity (SIROCCO), D. Krizanc and P. Widmayer, eds., Carleton Scientific, July 1997, pp. 162–175.

[23] C. Gavoille and N. Hanusse, *Compact routing tables for graphs of bounded genus*, in $26^{th}$ International Colloquium on Automata, Languages and Programming (ICALP), J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., vol. 1644 of Lecture Notes in Computer Science, Springer, July 1999, pp. 351–360.

[24] C. Gavoille and D. Peleg, *The compactness of interval routing for almost all graphs*, in $12^{th}$ International Symposium on Distributed Computing (DISC), S. Kutten, ed., vol. 1499 of Lecture Notes in Computer Science, Springer, Sept. 1998, pp. 161–174.

[25] 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.

[26] Y. Hassin and D. Peleg, *Sparse communication networks and efficient routing in the plane*, in $19^{th}$ Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, July 2000, pp. 41–59.

[27] L. Kleinrock and F. Kamoun, *Hierarchical routing for large networks; performance evaluation and optimization*, Computer Networks, 1 (1977), pp. 155–174.

[28] E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Log-space polynomial end-to-end communication*, SIAM Journal on Computing, 27 (1998), pp. 1531–1549.

[29] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, Second Edition, 1997.

[30] J. I. Munro and V. Raman, *Succinct representation of balanced parentheses, static trees and planar graphs*, in $38^{th}$ Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Oct. 1997, pp. 118–126.

[31] L. Narayanan and N. Nishimura, *Interval routing on $k$-trees*, Journal of Algorithms, 26 (1998), pp. 325–369.

[32] D. Peleg and E. Upfal, *A tradeoff between space and efficiency for routing tables*, in $20^{th}$ Annual ACM Symposium on Theory of Computing (STOC), Chicago, IL, May 1988, pp. 43–52.

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

[34] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5–8.

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