# A Theoretical Model for Routing Complexity

P. FRAIGNIAUD
*Université Paris Sud, France*

C. GAVOILLE
*Université Bordeaux I, France*

**Abstract**

This paper introduces a formal model for studying the complexity of routing in networks. The aim of this model is to capture both time complexity and space complexity. In particular, the model takes into account the input and output facilities of routers. A *routing program* is a RAM-program with five additional instructions that allow to handle incoming and outgoing headers, and input and output ports. One of these five additional instructions, called `release`, captures the possible use of hardware facilities to speed up routing.

Using our model, we show that there are routing functions which, if compacted, would require an arbitrarily large computation time to be decoded. The *latency* is the sum of the time (in bit-operation) required at every intermediate node to establish the route. We also show that, in any $n$-node network of diameter $D$, the latency is bounded by $O(D + n^{1/k} \log n)$, for every constant $k \geq 2$. This latter result has to be compared with the latency of the routing tables which is $\Theta(D \log n)$.

## 1 Introduction

The search for compact routing tables deserves more and more attention as the interest in large telecommunication systems grows [5]. Most of the literature devoted to compact routing focuses on strategies to encode point-to-point connections between pairs of sites in a networks. E.g., interval routing, introduced in [16, 17], is a famous compact routing method which encodes the set of destination addresses as a union of intervals. This method allows to decrease the memory requirement for routing in any network to $O(\log n)$ bits per communication link. However, if one insists on shortest paths, this method may fail, and may then require as many bits as the routing tables [8]. Therefore, other methods have been introduced whose goal is to satisfy some tradeoff between the space complexity of the routing tables on one hand, and the length of the routing paths on the other hand (see for instance [14]). Some of them use quite sophisticated techniques that allow to drastically reduce the memory requirement for routing, up to a small increase of the length of the routes [3], nay no increase at all [10] for particular types of networks.

However, compact routing protocols may suffer from a major drawback if reducing the space complexity of encoding the routes is up to the price of a large increase in the time complexity for computing these routes. This increase can be so high that reducing the length of the routing paths can be useless in front of the dominant per-hop computation time of the routers. This paper focuses on the tradeoff between space and time complexities for routing in networks. For that purpose, we develop a model to formally capture the two sides of the problem. Our model is based on the RAM-complexity enhanced with five additional instructions.

Using our model, we mainly show that there are routing functions that, if compacted, would require an arbitrarily large computation time to be decoded. We also show that, in any $n$-node network of diameter $D$, the latency, that is the sum of the time complexity (in bit-operation) required at each intermediate node to establish the route, is bounded by $O(D + n^{1/k} \log n)$, for every constant $k \geq 2$. This latter result has to be compared with the latency of the routing tables which is of $\Theta(D \log n)$.

# 2 Router, Routing Function, and Routing Program

## 2.1 The Network

We assume given a network of $n$ processing nodes, each with its own address, that is a bit string of arbitrary length. Each address is supposed to be unique in the network. Each node is connected to a *router*, that is a hardware chip dedicated to communications, in a one-to-one fashion. Note that connecting one processor to more than a single router would assume that processing nodes have some routing knowledge, whereas we want to specify that routing is a task dedicated to the routers. Similarly, connecting more than one processing node to a router is not considered in this paper since it changes the routing process locally only. Hence a network will be modeled by a graph $G = (V, E)$ where $V$ is the set of routers, and $E$ is the set of bidirectional links connecting the routers.

Each processing node is able to send and receive messages to and from any other node. A strong assumption that will hold all along this paper is that a node is not allowed to send a message to itself. The reason for this assumption is that a processor is supposed to have enough power to know its own address. From a more theoretical point of view, self-sending of messages causes many problems. For instance, it would be difficult to define the *stretch factor* [14] (that is the maximum, taken over all pairs of source-destination addresses, of the ratio between the length of the routing path between two nodes, over the distance between these two nodes in the network). Also, it would force a lower bound of $\Omega(\log n)$ bits for the local memory requirement for every networks, whereas we can hope to route with much less information locally.

A processing node is connected to its associated router by a bidirectional canal labeled 0, i.e., a canal that can be detected in $O(1)$ time. A message is composed of a set of data, plus a header containing information related to its destination. When a processor of address $x$ sends a message $M$ to another processor of address $y \neq x$, then the first header of $M$ is $y$. It means that the source of a message does not specify more information than the address of the destination. However, as we will see later, the router associated to the source can perform sophisticated computation on the destination address to generate more suitable headers.

For each router $x$, we denote by $\deg(x)$ the degree of $x$, that is the number of routers directly connected to $x$ by a bidirectional canal. The $\deg(x)$ input ports (resp. output ports) of $x$ are labeled locally from 1 to $\deg(x)$. When a message $M$ of header $h$ arrives in some router $x$ by its input port $p$, $0 \leq p \leq \deg(x)$, then $x$ computes a new header $h'$, and an output port $p'$. If $p' = 0$, then $M$ is forwarded to the corresponding processing node, and the routing ends. Otherwise, the header $h$ of $M$ is replaced by $h'$, and $M$ is forwarded on the output port $p'$.

The main subject of this paper is to model the processing task consisting, for every router, to compute the output pair $(h', p')$ from the input pair $(h, p)$. For this purpose, we need to formalize the behavior of a router.

## 2.2 A Router Model

In a router we can distinguish two parts according to the way the routing decision process is encoded. On the hardware side, there is the *switch*, whose role is to perform connections between the input ports and the output ports. The switch is any type of multistage network, or crossbar. It is controlled by the *Routing Control Processor* (RCP) which runs a software program that takes the routing decision as a function of the message headers and the input ports. The source of the routing program is encoded in the ROM (*Read Only Memory*) of the RCP, and each execution of the program makes use of RAM (*Random Access Memory*) available on the RCP.

When a message arrives to a router by some input port, its header $h$, and the label $p$ of the input port become available to the RCP. We do not want to specify where and how these two information are stored since it may depend on the used technology. We will just specify later the way the RCP has access to these information. The RCP program makes use of some bits of the input port and some bits of the header as input data to compute the output port and the (possibly) new header. Once the RCP has completed the computation of the output port, the corresponding switch connections are established between the input port and the selected output port, the new header is generated, and the message flows through the router.

The header and the input port label are supposed to be read-only data fields accessible by specific read instructions. The new header and the output port label are returned in specific data fields using particular types of write instructions. Each bit-address of the header and of the input port label is supposed to be readable only once. If the RCP needs multiple accesses to a bit of information contained in these fields, then the RCP must store this information in its RAM. Avoiding multiple accesses to the content of the header and to the content of the input port label prevents from the use of these data fields as variables for temporary storage. On the other hand, we allow some flexibility on the order in which the content (i.e., bits) of the input and output data fields can be accessed. Indeed specifying a particular order would be subject to technological constraints that are susceptible to be non valid in a near future.

An important point must be noticed. While the computation of the output port can be done quite efficiently in some networks (say in $O(1)$ or in $O(\log\log n)$ time), the sequential copy of the header onto the new header requires a time proportional to the size of the header at least. For two reasons, it would be unfair to put such a penalty for the time of a simple copy. From a theoretical point of view, such a penalty would not allow to consider sub-logarithmic routing complexities since $n$ nodes requires $\Omega(\log n)$ bits to be distinguished, and therefore headers of size $\Omega(\log n)$. As the model developed in [5], the goal is to reasonably charge the hardware in order to decrease the communication and time complexities for some distributed tasks. From a practical point of view, hardware copies can be easily performed by using dedicated physical supports that make the cost of a field-copy comparable to the cost of a bit RAM-instruction. Both reasons suggest that it is reasonable to charge $O(1)$ time for a simple input-output field-copy, as far as port and header fields are concerned. Furthermore, such a copy does not necessarily preserve the ordering. For instance, it is not more costly to copy one field to another in a reverse order than in the direct order. Actually, we will assume that any arbitrary permutation of the bits of the input fields is possible during the copy. On the other hand, we will put a strong restriction on the permutation: the used permutation must *not* depend of the values of the input port or of the header. This restriction is to prevent the use by the RCP of such permutations as computational facilities. Let us take an example to make clear why such assumptions.

**Example 1 (the odd-even routing)** *Let us consider routing in a directed network of in- and out-degree 2, typically a ring or a binary* de Bruijn *network (cf. Example 4). Let us label* even *and* odd *the two output ports of each node. Assume that the router of the source node computes the whole route to the destination node, and encodes the route into a bit-string which then form the first header. When an intermediate router receives a message, it reads the first bit of the header, takes the routing decision according to this bit. If it is a 0, then the message is routed on* *the even output port, otherwise it is routed on the odd output port. If the header is empty, the message is arrived at destination. The new header is formed from the header of the entering message by removing the first bit, the rest being identical. We want to get an $O(1)$ time-complexity for such a routing. This is the reason why, in our model, we charge $O(1)$ for the copy of the remaining bits of the former header to the new header.*

The next section formalizes the previous concepts.

## 2.3 Routing Function, and Routing Program

**Definition 1** *Let $G = (V,E)$ be a graph. A function $R : V \times \mathbf{N}^2 \to \mathbf{N}^2$ is a* routing function *on $G$ if there exists a labeling $\mathcal{L}$ of $V$ by integers and a labeling of the input and output ports of each node by integers such that, for any pair $(x,y)$ of nodes, $x \neq y$, there exists a sequence $(v_i,h_i,p_i,q_i) \in V \times \mathbf{N}^3$, $i = 0,\ldots,k$, $k \geq 1$, such that*

1. *$v_0 = x$, $v_k = y$, and $v_0,v_1,\ldots,v_k$ is a path from $x$ to $y$;*
2. *$p_0 = 0$, $q_k = 0$, and, for $i = 1,\ldots,k-1$, $q_i$ is the label of the output port of $v_i$ corresponding to the canal between $v_i$ and $v_{i+1}$, and $p_i$ is the label of the input port of $v_i$ corresponding to the canal between $v_{i-1}$ and $v_i$;*
3. *$h_0 = \mathcal{L}(y)$, and $(h_{i+1},q_i) = R(v_i,h_i,p_i)$, $0 \leq i \leq k$ (actually, $h_{k+1}$ is arbitrary).*

Informally, for any pair of nodes $(x,y)$, $x \neq y$, a message from $x$ to $y$ follows a path $x = v_0,v_1,\ldots,v_k = y$, entering $v_i$ through port $p_i$ with header $h_i$, and leaving $v_i$ through port $q_i$ with header $h_{i+1}$.
**Notation:** The distributed representation of $R$ is a collection of function $R_x : \mathbf{N}^2 \to \mathbf{N}^2$ defined by $R_x(h,p) = R(x,h,p)$.

The aim of this paper is to compute the time and space complexities required for routing. We have based our complexity model on the RAM model under logarithmic cost [2] with few additional I/O-instructions.

**Definition 2** *Let $G = (V,E)$ be a graph, and let $x \in V$ be any node of $G$. Given a routing function $R$ on $G$, a* routing program *for $R_x$ is a RAM-program computing the local routing function $R_x$ with the help of five additional statements:*

`read_header(i)` *This statement returns the ith bit of the header of the current message.*

`write_header(i,b)` *This statement writes the value $b \in \{0,1\}$ at the ith bit position of the new header.*

`read_port(i)` *This statement returns the ith bit of the input port label.*

`write_port(i,b)` *This statement writes the value $b \in \{0,1\}$ at the ith bit position of the output port label.*

$\texttt{release}_\pi$ *This instruction realizes the one-to-one function $\pi$ from the bits of the input header and input port label onto the bits of the output header and output port label which have not yet been set by previous* `write` *instructions.*

According to the logarithmic cost model the time complexity to compute `read_header(`$i$`)` is $O(\log i)$. The time to run $\texttt{release}_\pi$ for the RCP is $O(1)$, for every fixed $\pi$. In all the remaining of the paper, we denote by $\log n$ the logarithm of $n$ in base two.

**Example 2 (the *e*-cube routing)** *Let us consider the usual e-cube routing in hypercube [12]. Nodes of the d-dimensional hypercube $Q_d$ are labeled by integers from $0$ to $2^d - 1$, two nodes being adjacent if and only if the binary representation of their labels differ in exactly one place. Dimensions of the hypercube $Q_d$ are labeled from $1$ to d. The e-cube routing consists, upon reception of a message, to forward it in the dimension corresponding to the smallest bit position in which the destination address and the current node address differ. We give below a routing program implementing the e-cube routing in a node x.*

*Constants:*

$d \in \mathbf{N}$*; /* dimension of the cube */*
$k = \lceil \log(d+1) \rceil$*; /* size of the port labels */*
*x: array* $[1 \ldots d]$ *of* $\{0,1\}$*; /* address of the current node */*

*Variables:*

$i \in \{1,\ldots,k\}$;
$p \in \{0,\ldots,d\}$;
$b \in \{0,1\}$;

*Instructions:*

(1) $p := 0$;
(2) *For* $i := 1$ *to* $k$ *do* $p := 2 * p +$ `read_port(`$i$`)`;
(3) $b :=$ `read_header(`$p$`)`;
(4) *While* $b = x[p]$ *and* $p < d$ *do*
(5) $\quad p := p + 1$;
(6) $\quad$ `read_header(`$p$`)`;
(7) *If* $b = x[p]$ *then* $p := 0$;
(8) *For* $i := 1$ *to* $k$ *do*
(9) $\quad$ `write_port(`$i, p \bmod 2$`)`;
(10) $\quad p := p \div 2$;
(11) $\texttt{release}_{Identity}$.

*In the above routing program, b is a bit, p, i, k, and d are integers, and x is an array of bits. Parameters x, d, and k are constants of the program. They denote the address of the current node, the dimension of the hypercube, and the length* *of the binary representation of d (plus one in order to distinguish the local host), respectively. The two first instructions simply compute the input port label p from its available binary expression. Instruction (3) extracts the value of the pth bit of the header. Instructions (4) to (6) look for the smallest bit-position (greater than p) in which the destination address and the current address differ. Instruction (7) sets the output port to 0 if the destination address and the current address does not differ. Instructions (8) to (10) write the output port value. Instruction (11) copies the entire header to get the new header. The model charges $O(1)$ time for this last instruction.*

**Remark:** Since we focus on bit complexity of the routing program, we have refer to the *logarithmic cost criterion* [2] when computing the time complexity of the routing algorithm. It implies that the allowed arithmetical operations are only those whose complexity is linear as a function of the size of the inputs (addition, shift, etc.).

## 3 ROM and RAM

Given the local expression $R_x$ on node $x$ of a routing function $R$ on a graph $G$, a pair of integers $(h,p)$ is a valid entry for $R_x$ if, while using $R$ on $G$, a message of header $h$ can indeed enter the router $x$ on port $p$.

**Definition 3** *Let $P_x$ be a routing program encoding a routing function $R_x$ on node $x$ of a graph $G$. We define:*

- $\text{ROM}(P_x)$ *as the length in bits of the routing program $P_x$.*
- $\text{RAM}(P_x)$ *as the maximum, taken over all valid entries $(h,p)$, of the bit memory space required by the registers of $P_x$ during its execution on $(h,p)$.*

Note that constants are a part of $P_x$. The length of an instruction $\texttt{release}_\pi$ is $O(1)$, for every fixed $\pi$. If $P_x$ is written with $k$ different release instructions, $\texttt{release}_{\pi_1}, \ldots, \texttt{release}_{\pi_k}$, then each instruction $\texttt{release}_{\pi_i}$ has length $O(\log k)$. However, in all the remaining of the paper we will consider $k$ to be small constant.

Let us consider the routing program $P_x$ of Example 2 which encodes $e$-cube routing on the hypercube $Q_d$ of $n = 2^d$ nodes. Since $O(\log n)$ bits are enough to encode all constants $d$, $k$ and $x$, and since the program itself consists of $O(1)$ instructions, we get that $\text{ROM}(P_x) = O(\log n)$. Similarly, since $p$ and $i$ are of maximum size $O(\log\log n)$, and since $b$ is of size $O(1)$, we get that $\text{RAM}(P_x) = O(\log\log n)$.

**Example 3 (routing tables)** *Let us consider any graph G. Nodes are arbitrary labeled by integers from $1$ to n, and input and output ports from $1$ to $\deg(x)$. Each* *node x stores a table $T_x$ of n entries of $O(\log \deg(x))$ bits. The entry y specifies the output port $T_x[y]$ on which the message must be forwarded to get to destination y. A simple routing program $P_x$ shows that $\text{ROM}(P_x) = O(n \log \deg(x))$, and $\text{RAM}(P_x) = O(1)$. Indeed, the bits of $T_x[y]$ can be read from the memory, and written online in the output port, without intermediate storage in any RAM registers. The* `release` *instruction is used here to copy the header without any modification of its content.*

**Definition 4** *Let $R_x$ be the local expression on node x of a routing function R on a graph G. We define:*

- $\text{ROM}(R_x) = \min_{P_x} \text{ROM}(P_x)$;
- $\text{RAM}(R_x) = \min_{P_x} \text{RAM}(P_x)$;

*where the minimization is performed on all routing programs of $R_x$.*

Of course, $\text{ROM}(R_x)$ is strongly related to the Kolmogorov complexity [13] of $R_x$, that is the length of the smallest Turing Machine program that computes $R_x$. Note however that these two notions slightly differ, mainly because of the instruction `release` allowing to print many bits of the output in constant time. Informally, let us denote by $C(R_x)$ the length of the shortest RAM-program which computes $R_x(h, p)$ for all valid entries $(h, p)$. $C(R_x)$ is therefore the Kolmogorov complexity of $R_x$, up to an additive constant (see "The Invariance Theorem", [13]). The following relation holds:

**Lemma 1** *Let $R_x$ be the local expression on node x of a routing function R on a graph G. Let $\ell_i$ (resp. $\ell_o$) be the maximum size of the header of a message entering x (resp. leaving x). Let $\ell_p = \log \deg(x)$. Then,*

$$\text{ROM}(R_x) > C(R_x) - O\left((\ell_i + \ell_p)\log(\ell_o + \ell_p)\right).$$

**Proof.** We have to consider the power of the function `release`. Let us simulate a `release` instruction by a RAM-program instruction sequence. Each bit of the input fields (incoming header and input port) can be copied onto the output fields (outgoing header and output port) using the function `release`. However, it is not allowed to copy a bit simultaneously to several output positions. The total number of possible copies from a field of bits, $X$, of size $x$ to a field $Y$ of size $y$ is $(y+1)^x$ (as each bit of $X$ has $y$ ways to be copied in $Y$, plus one if it is not copied). Let $\ell_p$ be the size of the labels of in/output ports. $\ell_p \leq \log \deg x + O(1)$. There are $\alpha = (\ell_o + \ell_p + 1)^{\ell_i + \ell_p}$ ways of doing the copies. So, $\log \alpha + O(1)$ bits are enough to encode the `release` instruction by a sequence or read/write instruction between the input and the output fields. □

For headers of size $O(\log n)$, the previous lemma shows that $\text{ROM}(R_x) \geq C(R_x) - O(\log n \log\log n)$. The difference between the ROM complexity and the

Kolmogorov complexity is due to the `release` instruction that can be used to shorten RAM-programs by transformation into routing programs. The reduction can be significant when messages of large headers are used. Note that headers received on port 0 (from the host) consist of destination address only. This allows to lower bound the ROM parameter even for routing functions that may use headers of unbounded size (as it can be the case when the whole route is computed locally on the first router, to be then encoded in the header). Therefore, Lemma 1 can be used to get the same lower bounds for the ROM parameter as all lower bounds given in the literature related to the incompressibility of the routing tables [4, 7, 9, 11].

In general, we are interested in finding the "best" routing function for a given graph $G$ among a class of routing functions. For instance we would like to find a shortest path routing function $R$ on a graph $G$ such that $\text{ROM}(R_x)$ and $\text{RAM}(R_x)$ are both minimum for all nodes. This is the motivation of the following definition:

**Definition 5** *Let $G = (V,E)$ be any graph, let $x \in V$, and let $\mathcal{R}$ be a class of routing functions on G. We define:*

- $\text{ROM}(x,\mathcal{R}) = \min_{R_x \in \mathcal{R}} \text{ROM}(R_x)$*;*
- $\text{RAM}(x,\mathcal{R}) = \min_{R_x \in \mathcal{R}} \text{RAM}(R_x)$.

For a practical point of view, ROM and RAM must be balanced in order to satisfy the compactness requirement. For instance, for $\mathcal{R}^* = \{$ shortest path routing functions $\}$, it was shown in [10] that, for every $n$-node Cayley graph, $\text{ROM}(x,\mathcal{R}^*) = O(\log^3 n)$, for any node $x$. This result is based on a nice $O(\log^3 n)$ bits encoding of the multiplicative tables of the groups. Using this information, each router is able to reconstruct the entire network, and therefore to compute all shortest paths. However such a somewhat brute force routing program $P_x$ needs to rebuild the adjacency matrix of the graph in the router's memory. Therefore, such a routing may require $\text{RAM}(P_x) = O(n^2)$ bits.

Actually, the use of a routing tables allows to give a better trade-off between ROM and RAM, namely $\text{RAM}(x,\mathcal{R}^*) = O(1)$ and $\text{ROM}(x,\mathcal{R}^*) = O(n \log n)$ for every node $x$ (cf. Example 3).

The tradeoff between ROM and RAM is considered in the next section.

# 4 SPACE and TIME

**Definition 6** *Let $P_x$ be a routing program on node $x$ of a routing function $R$ on a graph $G$. We define:*

- $\text{SPACE}(P_x) = \text{ROM}(P_x) + \text{RAM}(P_x)$*, and*
- $\text{TIME}(P_x)$ *as the maximum, taken over all valid entries $(h,p)$, of the number of bit-operations executed by $P_x$ on $(h,p)$, the* `release` *instruction being considered as one bit-operation.*

Note that, as already stated before, it is natural to fix the cost of `release` as a constant. Indeed, `release` can be implemented using hardware features whose complexity can be considered as negligible in front of software instructions, even if the size of the circuit encoding `release` increases with the size of the network. Actually, the depth of the circuit implementing `release` is really a constant independent of the size of the network.

**Definition 7** *Let $R_x$ be the local expression on node $x$ of a routing function $R$ on a graph $G$. We define:*

- $\text{SPACE}(R_x) = \min_{P_x} \text{SPACE}(P_x)$*;*
- $\text{TIME}(R_x) = \min_{P_x} \text{TIME}(P_x)$*;*

*where $P_x$ is any routing program of $R_x$. Moreover, for any class $\mathcal{R}$ of routing functions on $G$, we define:*

- $\text{SPACE}(x, \mathcal{R}) = \min_{R_x \in \mathcal{R}} \text{SPACE}(R_x)$*;*
- $\text{TIME}(x, \mathcal{R}) = \min_{R_x \in \mathcal{R}} \text{TIME}(R_x)$*.*

**Notation:** For every graph $G$, we denote $\text{SPACE}(G) = \min_{R_x \in \mathcal{R}} \max_x \text{SPACE}(R_x)$, and $\text{TIME}(G) = \min_{R_x \in \mathcal{R}} \max_x \text{TIME}(R_x)$, i.e., unless specified, only shortest path routing functions are considered.

Applying $e$-cube routing on $Q_d$, we get $\text{SPACE}(Q_d) = O(\log n)$. Moreover, all instructions takes at most $O(\log d)$ bit-operations to be performed, and loop (4) is executed at most $d$ times. So, $\text{TIME}(Q_d) = O(d \log d) = O(\log n \log\log n)$.

Clearly, routing tables (Example 3) show that every graph $G$ of maximum degree $d$ satisfies $\text{SPACE}(G) = O(n \log d)$ and $\text{TIME}(G) = O(\log n)$. Recall that, in the logarithmic cost criterion model, reading $O(\log d)$ bits in a table of size $O(n \log d)$ takes $O(\log (n \log d) + \log d)$ bit-operations.

The next theorem shows that there is a tradeoff between compact routing and time-efficient routing.

**Theorem 1** *Let*

$$s(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$$

*be a constant size stack of 2's powers. For infinitely many $n$, there exists a shortest path routing function $R$ on an undirected $n$-node graph $G$ such that $\text{ROM}(R_x) = O(\log n)$ for every node $x$, and such that there exists a node $x_0$ for which any routing program $P_{x_0}$ of $R_{x_0}$ satisfying $\text{ROM}(P_{x_0}) < n - \log^2 n$ also satisfies $\text{RAM}(P_{x_0}) > s(n)$.*

**Proof.** Let $A \subset \mathbf{N}^*$ be a suitable set, and let $G = (V, E)$ be a path of $n$ nodes. We set $A_n = \{i \in A \mid i \leq n\}$, and $\bar{A}_n = \{i \notin A \mid i \leq n\}$. We set $A_n = \{a_1, \ldots, a_k\}$,

$a_1 < \cdots < a_k$, $k \geq 0$, and $\bar{A}_n = \{\bar{a}_1, \ldots, \bar{a}_l\}$, $\bar{a}_1 < \cdots < \bar{a}_l$, $l \geq 0$. We identify the nodes with their labels as follows: $V = \{1, \ldots, n\}$, and $E = E_1 \cup E_2$, with $E_1 = \{(\bar{a}_i, \bar{a}_{i+1}) \mid i < l\} \cup \{(\bar{a}_l, a_1)\} \cup \{(a_j, a_{j+1}) \mid j < k\}$, and, symmetrically $E_2 = \{(x,y) \mid (y,x) \in E_1\}$. The arcs of $E_1$ correspond to the output ports labeled 1, and arcs of $E_2$ with the output ports labeled 2. Let $R$ be the unique shortest path routing function on the path. Let $C(X)$ be the Kolmogorov complexity of $X$, that is the length of the smallest program that prints $X$ and halts.

**Claim 1** $\textsc{Rom}(R_x) \leq C(A) + O(\log n)$, *for every* $x \in V$.

**Proof.** $R_x$ can be implemented as follows: given a destination $y$, the router $x$ compares $y$ to its own labels, and returns the output port 0 if equals. It returns the output port 1 if either $x \in \bar{A}_n$ and $y \in A_n$, or if $x < y$, and $x, y$ are both in $\bar{A}_n$ or both in $A_n$. Node $x$ returns port 2 otherwise. The header is copied without any change using the `release` instruction. Such a routing program routes along shortest paths, and it can be described from the set $A$, and the integers $x$, $a_1$, $\bar{a}_l$, and $n$. Hence its $\textsc{Rom}$ is at most $C(A) + O(\log n)$. □

For any infinite binary string $\chi$, let $\chi_{1:n}$ denotes the $n$ first bits of $\chi$. Let $C^t(X)$ be the Kolmogorov complexity of $X$ with time resource bounded by $t$, i.e., the length of the shortest program that prints $X$ and halts in time at most $t$. $C^t(X|n)$ denotes $C^t(X)$ in which $n$ is known. We have $C^t(X|n) \leq C^t(X)$. By Theorem 7.4 in [13, page 384], there is a recursive binary sequence $\chi$ such that $C^t(\chi_{1:n}|n) \geq n - f(n)$ infinitely often, where $t$ and $f$ are two arbitrary unbounded total recursive functions.

Hence, let us define $A$ as the unique set whose $\chi$ is the characteristic sequence. $A$ is recursive enumerable. By Barzdin's Lemma [13, page 138], $C(A) = O(\log n)$, and hence by Claim 1, $\textsc{Rom}(R_x) = O(\log n)$ for every $x \in V$. Now, let $x_0 = \bar{a}_l$. For $n$ large enough, $x_0 \in V$.

**Claim 2** $C^t(R_{x_0}) \geq C^{nt}(A_n) - O(\log n)$.

**Proof.** By definition of $R$, from the knowledge of $R_{x_0}$ and $n$, one can rebuild $A_n$ in time $nt$ because $y \in A_n$ if and only if $R_{x_0}(y,0) = (y,1)$, i.e., if and only if $x_0$ forwards $y$ onto port 1 at the first step. So, $C^{nt}(A_n) \leq C^t(R_{x_0}) + O(\log n)$. □

Assume the routing program $P_{x_0}$ satisfies $\textsc{Time}(P_{x_0}) \leq t$. Applying Lemma 1, $\textsc{Rom}(P_{x_0}) \geq C^t(R_{x_0}) - O(\log n \log\log n)$ because headers are of size $O(\log n)$. By Claim 2,

$$\textsc{Rom}(P_{x_0}) \;\geq\; C^{nt}(A_n) - O(\log n \log\log n).$$

By Theorem 7.4 of [13, page 384], $C^{nt}(A_n) \geq C^{nt}(A_n|n) \geq n - f(n)$ infinitely often. Finally, $\textsc{Rom}(P_{x_0}) \geq n - \log^2 n$, for a suitable total recursive function $f(n) = \Theta(\log^2 n)$, and for $n$ large enough.

Moreover, every (finite and converging) routing program $P$ satisfies $\mathrm{RAM}(P) \geq \log \mathrm{TIME}(P)$ (otherwise the running of the program would enter twice in the same state, and therefore would not converge). Let us fix $t = 2^{s(n)}$. The function $t$ is an unbounded total recursive function. Thus if $\mathrm{ROM}(P_{x_0}) < n - \log^2 n$, then $\mathrm{TIME}(P_{x_0}) > t$, and therefore

$$\begin{aligned} \mathrm{RAM}(P_{x_0}) &\geq \log \mathrm{TIME}(P_{x_0}) \\ &> \log t \\ &> s(n) \end{aligned}$$

and the proof is completed. □

Since $\mathrm{TIME}(P_x) \geq \mathrm{RAM}(P_x)$, for every $P_x$, it follows that TIME might be exponentially large when ROM is optimal. As an open problem, we ask whether there is a graph for which, for every labeling of $G$ and every shortest path routing function $R$, there is a node $x$ for which $\mathrm{ROM}(R_x) = O(\mathrm{ROM}(x, \mathcal{R}^*))$ implies $\mathrm{RAM}(R_x) > s(n)$. Unfortunately, the example of Theorem 1 fails is this case as we have forced a particular labeling of the nodes.

# 5 LATENCY

$\mathrm{TIME}(G, \mathcal{R})$ may not reflect the real time of delivery of messages in $G$, even if one takes into account the length of the longest path generated by $R \in \mathcal{R}$. Indeed, the worst case of the local routing time complexity is not necessarily achieved at all nodes of a routing path.

**Example 4 (the de Bruijn routing)** *The typical example is the binary* de Bruijn *network of dimension $d$ [15], denoted by $B_d$. Nodes of $B_d$ are the $2^d$ binary words of length $d$, and there is an edge from $x_d x_{d-1} \ldots x_2 x_1$ to the two nodes $x_{d-1} \ldots x_2 x_1 \lambda$ where $\lambda \in \{0,1\}$. Routing in $B_d$ can be achieved as follows: the source computes the maximum length $k$ of a prefix of the destination address that is a suffix of the source address, and the header is formed by corresponding prefix of length $k$. Upon reception of a message, an intermediate router considers the first bit of the header, and routes according to this bit (if the header is empty, the message is arrived at destination). The new header is obtained from the incoming header by removing the first bit. It guarantees a routing along the shortest paths. It takes $O(d)$ arithmetic operations to compute the original header (cf. Boyer-Moore pattern matching algorithm [1]). Each arithmetic operation is performed on integers of size $O(\log d)$. Hence it takes $O(d \log d) = O(\log n \log\log n)$ bit-operations to compute the header. Each intermediate router routes in constant time. Therefore $\mathrm{TIME}(B_d) = O(\log n \log\log n)$, whereas the time for a message to be routed from its source to its destination is $O(\log n \log\log n + D)$ that is also $O(\log n \log\log n)$ since the diameter of the* de Bruijn *network is $D = \log n$.*

The previous example motivates the LATENCY parameter:

**Definition 8** *Let $G = (V,E)$ be any graph, and let $R$ be a routing function on $G$. Let $R_{x\to y}$ denote the routing path from $x$ to $y$ induced by $R$. Given a routing program $P_u$ of $R_u$ for every $u \in R_{x\to y}$, let us denote by $t_{x\to y}(P_u)$ the time (in bit-operation) to compute $P_u$ when $R_{x\to y}$ cuts the node $u$.*

$$\text{LATENCY}(R) \;=\; \min_{P} \max_{x\neq y\in V} \sum_{u\in R_{x\to y}} t_{x\to y}(P_u)$$

*where the minimization is performed on all collections $P$ of routing programs for $R$. Moreover, for every class $\mathcal{R}$ of routing functions on $G$, we define:*

$$\text{LATENCY}(G,\mathcal{R}) \;=\; \min_{R\in\mathcal{R}} \text{LATENCY}(R)$$

**Notation:** Unless it is clearly stated, $\text{LATENCY}(G)$ denotes $\text{LATENCY}(G,\mathcal{R}^*)$ where $\mathcal{R}^*$ denotes the class of shortest path routing functions on $G$.

$E$-cube routing on $Q_d$ gives $\text{LATENCY}(Q_d) = O(\log n \log\log n)$, that is strictly smaller than $\log n\, \text{TIME}(Q_d)$. Indeed, due to instruction (3) in Example 2, the loop (4) is globally executed at most $\log n$ times along the path between any two nodes of $Q_d$. Therefore, the contribution of this loop to the latency is of $O(\log n \log\log n)$. On the other hand, instructions (2) is executed at every intermediate nodes of a routing path. The cost of instruction (2) is $O(\log\log n)$, and thus the latency is $O(\log n \log\log n)$.

It is quite easy to check that, for products of simple graphs like paths, or rings, we get a latency smaller than $D$ times the TIME of the nodes, where $D$ is the diameter of the graph. Since $\text{TIME}(G) = O(\log n)$ by the use of routing tables (Example 3), we get that $\text{LATENCY}(G) = O(D \log n)$ for an arbitrary graph $G$. Actually, it is possible to get a lower latency for large diameter as it is shown in the following theorem.

**Theorem 2** *For every $n$-node graph $G$ with node labels $V = \{1,\dots,n\}$ and of diameter $D$, and for every constant $k \geq 2$, there exists a routing function $R$ that respects $V$ (name-independent) with headers of size $O(\log n)$ such that $\text{LATENCY}(R) = O(D + n^{1/k} \log n)$.*

**Proof.** The way to build $R$ is inspired of [17] where a covering of graphs by sets of radius bounded by $O(n^{1/k})$ is presented (similar decompositions can be founded in [6]). The centers of the set are connected altogether with a spanning tree $T$. Each set, or region, is then recursively decomposed in $O(k)$ levels. Messages are routed along shortest paths from a given level to a higher level, until they reach a center. Then, messages are routed along $T$, and, eventually, they go down towards their destinations along shortest paths inside the region of the destination.

In [17], it is shown that the total number of "routing decisions" along a route is at most $O(kn^{1/k})$. More precisely, a routing decision means an access to a table depending on the destination. A router makes "no routing decision" if the selected output port is computed independently of the destination. However, it does not implies that such routers consume $O(1)$ time for the routing. There is no decision whenever a message goes up towards the center of the its region of origin, whereas there is one decision every time a center is reached in $T$, and at every node during the downward phase (in particular to check if the message reached its destination). The current phase can be set as part of the header with a field of $O(\log k)$ bits size. Each time the message enters a new level, or a center node, an access to a full routing table is performed. Such an access requires $O(\log n)$ bit-operations. A message together with its phase number in its header, say phase $i$, arriving at a given node, is forwarded to the link labeled $p_i \leq i$. $p_i$ can be tabulated so that the routing program runs in time $O(\log k)$ to read and write $p_i$. (The non-written bits of the output port field can be assumed to be reseted with a suitable `release` instruction.) The length of the route is bounded by $O(kD)$. In total the latency of $R$ is bounded by:

$$O(kD \log k) + O(kn^{1/k} \log n)$$

which is, for $k = O(1)$ and $n$ large enough, bounded by $O(D + n^{1/k} \log n)$. Note that for every node $x$, $\textsc{Time}(R_x) = O(\log n)$ and $\textsc{Rom}(R_x) = O(n \log n)$ since routing tables are used for the centers. □

# 6 Conclusion

By the use of routing tables, and as far as shortest path routing is concerned, we have shown that we can simultaneously guarantee SPACE ($= \textsc{Rom} + \textsc{Ram}$) in $O(n \log n)$, and TIME in $O(\log n)$ for each graph. Moreover, we have shown that there are routing functions for which is not possible to optimize both ROM and TIME criteria. However we were unable to prove or disprove that for every graph it is possible to route with optimal SPACE and optimal TIME, up to a multiplicative constant. More formally, we propose the following question as an open problem.

**Problem 1** *For every n-node graph G, does there exist a (shortest path) routing function R satisfying that for every node x of G, there exist a routing program $P_x$ implementing $R_x$ on x, such that:*
*1)* $\textsc{Space}(P_x) = O(\textsc{Space}(G))$ *and* $\textsc{Time}(P_x) = O(\textsc{Time}(G))$*?*
*or*
*2)* $\textsc{Space}(P_x) = O(\textsc{Space}(G))$ *and* $\textsc{Time}(P_x) = O(\log^k n)$*, for a constant k?*

# References


[1] AHO, A. V. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science* (1990), J. van Leeuwen, Ed., vol. A, North-Holland, pp. 255–300.

[2] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] AWERBUCH, B., BAR-NOY, A., LINIAL, N., AND PELEG, D. Improved routing strategies with succint tables. *Journal of Algorithms 11* (Feb. 1990), 307–341.

[4] BUHRMAN, H., HOEPMAN, J.-H., AND VITÁNYI, P. Space-efficient routing tables for almost all networks and the incompressibility method. *SIAM Journal of Computing* (1998). To appear.

[5] CIDON, I., GOPAL, I., AND KUTTEN, S. New models and algorithms for future networks. In $7^{th}$ *Annual ACM PODC* (Aug. 1988).

[6] ERDÖS, P., GERENCSÉR, L., AND MÁTÉ, A. Problems of graph theory concerning optimal design. In *Colloq. Math. Soc. Janos Bolyai* 4*: Combinatorial theory and its applications* (1970), P. Erdös, A. Renyi, and V. T. Sós, Eds., vol. 1, North-Holland, pp. 317–325.

[7] FRAIGNIAUD, P., AND GAVOILLE, C. Universal routing schemes. *Journal of Distributed Computing 10* (1997), 65–78.

[8] GAVOILLE, C., AND GUÉVREMONT, E. Worst case bounds for shortest path interval routing. *Journal of Algorithms 27* (1998), 1–25.

[9] GAVOILLE, C., AND PÉRENNES, S. Memory requirement for routing in distributed networks. In $15^{th}$ *Annual ACM PODC* (May 1996), ACM PRESS, Ed., pp. 125–133.

[10] KRANAKIS, E., AND KRIZANC, D. Boolean routing on Cayley networks. In $3^{rd}$ *International Colloquium SIROCCO* (June 1996), N. Santoro and P. Spirakis, Eds., Carleton University Press, pp. 119–124.

[11] KRANAKIS, E., AND KRIZANC, D. Lower bounds for compact routing. In $13^{th}$ *Annual STACS* (Feb. 1996), C. Puech and R. Reischuk, Eds., vol. 1046 of Lecture Notes in Computer Science, Springer-Verlag, pp. 529–540.

[12] LEIGHTON, F. T. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. Morgan Kaufmann, 1992.

[13] LI, M., AND VITÁNYI, P. M. B. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, 1993.

[14] Peleg, D., and Upfal, E. A trade-off between space and efficiency for routing tables. *Journal of the ACM 36*, 3 (July 1989), 510–530.

[15] Pradhan, D. K., and Samathan, M. R. The de Bruijn multiprocessor network: A versatile sorting network. In *IEEE Proceedings of the* $12^{th}$ *International Symposium on Computer Architecture* (June 1985), pp. 360–367.

[16] Santoro, N., and Khatib, R. Labelling and implicit routing in networks. *The Computer Journal 28*, 1 (1985), 5–8.

[17] van Leeuwen, J., and Tan, R. B. Computer networks with compact routing tables. In *The Book of L* (1986), G. Rozemberg and A. Salomaa, Eds., Springer-Verlag, pp. 259–273.

**P. Fraigniaud** is with LRI – CNRS, Université Paris Sud, 91405 Orsay Cedex, France.

**C. Gavoille** Université, LaBRI, Université Bordeaux I, 351, cours de la Libération, 33405 Talence Cedex, France