Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux I, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report `RR-1252-01`

Routing in Trees

by Pierre Fraigniaud and Cyril Gavoille

January, 2001

Routing in Trees

Pierre Fraigniaud* Cyril Gavoille†

Abstract

This article focuses on routing messages along shortest paths in tree networks, using compact distributed data structures. We mainly prove that n-node trees support routing schemes with message headers, node addresses, and local memory space of size $O(\log n)$ bits, and such that every local routing decision is taken in constant time. This improves the best known routing scheme by a factor of $O(\log n)$ in term of both memory requirements and routing time. Our routing scheme requires headers and addresses of size slightly larger than $\log n$, motivated by an inherent trade-off between address-size and memory space, i.e., any routing scheme with addresses on $\log n$ bits requires $\Omega(\sqrt{n})$ bits of local memory-space. This shows that a little variation of the address size, e.g., by an additive $O(\log n)$ bits factor, has a significant impact on the local memory space.

Keywords: compact routing, trees, routing algorithms.

1 Statement of the 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*, that is a mechanism working in a distributed fashion, and allowing source-to-destination messages delivery in the network, for any source and any destination.

Quite often, the design and the optimization of the management and control systems of the network, including routing, is done after the network construction. In this case, the routing problem is stated as follows: given a graph (i.e., the underlying topology of a communication network), design for each node of the graph (i.e., for each router of the network) a efficient computable function that determines, for every message entering a node, the link on which the message has to be forwarded. This computation is performed as function of the information contained in the header of the message, e.g., the destination

*CNRS, Laboratoire de Recherche en Informatique, Université Paris-Sud, 91405 Orsay cedex, France. *pierre@lri.fr*. Part of this work was done while the first author was visiting Carleton University at Ottawa, and Concordia University at Montréal. Additional support from Carleton U, the NATO, and the RNRT project R.O.M.

†Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux I, 351, cours de la Libération, 33405 Talence Cedex, France. *gavoille@labri.u-bordeaux.fr*.

node address, and of local information stored at the node. The term "efficient" groups a set of desirable quality factors like: routes of reasonable length (compared to shortest paths); low local computation time; limited link-congestion; compact local data structures; fault tolerance; etc.

We want to point out that the complexity results on routing are quite strongly dependent of the model, including, e.g., the ability to setup the addresses of the nodes, the ability to modify the headers, etc. Moreover, we also want to stress that results strongly differ according to slight variations of parameters, including, e.g., the size of the header, the length of the routes, etc. Therefore, let us make clear the framework of this paper.

In the remaining, the communication network is denoted by a connected and undirected n-node graph $G = (V, E)$. The routing scheme makes use of *addresses*, every node being identified by an address which is unique in the network. Hereafter, the address of node u is denoted by $\ell(u)$. The routing scheme makes also use of a *routing algorithm* that is a distributed algorithm whose role is to route messages. Upon reception of a message by node u, the routing decision taken at u is performed by a *routing function*. It depends solely on (1) the message header, (2) the incoming link, and (3) local data structures stored at u. The design of the routing scheme includes not only de design of the routing algorithm via the definition of the routing function, but also the setting of the address-set, the header-set, and of the local data-structures. Moreover, it has to incorporate the link labeling.

More precisely, incoming and outgoing links from every node u are identified by numbers from 1 to $\deg(u)$, the degree of u. This labeling is local, and should not necessarily be consistent between neighbors. In particular, the edge $\{u, v\}$ can be labeled i by u, and j by v, with $i \neq j$. Each node (i.e., each router) is also supposed to have an extra link, connecting it to its local host. This link is labeled 0. Once the routing function has selected the outgoing link on which the message must be forwarded, i.e., a number between 0 and $\deg(u)$, it creates a new header for that message, and it sends the message with its header through that link. The original header of a message is given by the source host; it simply consists of the address of the destination, and, upon reception of a message from the source host, the attached router may modify the header, but the result of this modification depends solely on the destination address, and, again, local data structures stored by the router. We point out here that no time or space consuming computations are hidden in this model.

A specific type of routing is called *direct*. Direct routing imposes that the message header is fixed at once by the source host, and cannot be modified by intermediate nodes on its route toward the destination. In other words, the header set is exactly the node address set. Direct routing schemes are desirable because they do not require extra hardware facilities to rewrite headers. Note though that relaxing this constraint allows to construct sophisticate routing schemes for arbitrary graphs (cf., e.g., the hierarchical routing schemes introduced in [1, 10]).

To summarize, solving the routing problem consists to design the whole routing scheme, including (1) the routing function, (2) the format of the addresses, (3) the local labeling of the links, (4) the format of the headers, and (5) the local information to be stored locally at

each node in order to perform the computation of the routing function. The complexity of the setting-up of these data is measured by the time it takes to perform those computations from given the graph G, and is refered as the *pre-processing time*. The other parameters that are typically addressed in the literature are the following. The *size of the addresses* (that is also the *size of the headers* for direct routing schemes) is measured by the range of the set in which these addresses are taken. More precisely, for $m \geqslant n$, a routing scheme is of $[1, m]$-range address if the addresses are taken from the set $\{1, 2, \ldots, m\}$. (Alternatively, such a scheme is called a $\log m$-bit address routing scheme.) The *local memory space* is the size of all the data structures required by each node to compute the routing function. The *routing time* is the worst-case time-complexity of the routing function to perform its computation. The *length of the routes* is measured by competition against the shortest paths.

In this paper we consider shortest-path routing only. Moreover, all the routing schemes we will construct are direct, i.e., routing will be performed based on the destination address only, and headers are not modified once set by the source. Regarding the local labeling of the links, we will consider two main variations of the problem: the *designer-port* model, and the *fixed-port* model. The former allows the designer of the routing scheme to freely label (between 1 and $\deg(u)$) the incident links of each node u while constructing and optimizing the data structures of the routing scheme. On the contrary, the latter assumes that the local labeling of the links is given is advance. The fixed-port model forces the use of correspondence tables if the routing function needs a specific labeling to perform efficiently, and therefore this model may lead to a significant increase of the size of the data structures compared to the designer-port model. On the positive side though, the fixed-port model allows to superpose several routing schemes such as, e.g., hierarchical routing schemes [1, 10].

Before stating the results included in this paper, we list some previous and related works.

Related works

One of the most widely used routing scheme for arbitrary graphs is the *routing tables*. It consists to label the nodes from 1 to n, and to store an n-entry table at each node, such that its i-th entry returns the local number of the adjacent link on which a message of destination address i has to be transmitted. Routing tables are direct and use $[1, n]$-range addresses, require $O(n \log d)$ bits of memory at each node of degree d in the fixed-port model, and perform routing in $O(1)$ time at each node. Although $O(n \log d)$ memory bits seems quite high, it must be pointed out that this is the best that can be achieved for arbitrary graph. More precisely, it has been proved in [5] that $\Omega(n \log d)$ bits is a lower bound for shortest-path routing schemes using addresses on $c \log n$ bits, for every constant $c \geqslant 1$. This lower bound holds in the designer-port model. According to this result, a tremendous effort has been done to derive routing schemes requiring smaller memory space for specific classes of networks, in particular for trees.

In trees, a common strategy consists to set the addresses via a Depth-First Search (DFS)

traversal of the tree, and then to use the *interval routing scheme* (IRS) of [11, 12] (see [4] for a survey on IRS). More precisely, at every internal node u of the tree, the edge leading to a child v is associated to an interval containing the set of the addresses of the nodes in the subtree rooted at v, and the link leading to the root is associated to the complementary interval (this latter interval may wrap modulo n). The memory space is $O(d \log n)$ bits at each node of degree d in the fixed-port model, and the use of a binary search in the set of intervals provides a $O(\log n)$ routing time at each node. The total amount of routing informations, summed over all nodes, is low compared to the routing tables, i.e., $O(n \log n)$ bits in total. However, IRS does not evenly balance routing information among the nodes since $O(n \log n)$ bits could be required locally for trees of large degree.

Cowen [2] proposed a direct routing scheme in the fixed-port model for trees of large degree using $O(\sqrt{n} \log n)$ bits of memory, $3 \log n$ bits of addresses, and $O(\log n)$ routing time. The address-space can be optimized considering the designer-port model: the direct routing scheme in [3] uses $O(\sqrt{n} \log n)$ bits of memory with $[1, n]$-range addresses, and $O(\log n)$ routing time. Actually, if we insist on small memory space, [4] shows that there exists a routing scheme for trees with $[1, n]$-range addresses, $\alpha\sqrt{n}$ bits of memory, where $\alpha \approx 3.7$. However, this latter scheme may involve exponential routing times.

Peleg [9] proposed the best known routing scheme for trees, inspired from the so-called "distance labeling" schemes, and stated in the fixed-port model. It uses $O(\log^2 n)$ bits for both the addresses and the memory at each node, and requires $O(\log n)$ routing time. Actually, the routing algorithm is direct and based on the destination-address only so that the node needs only to store its address. That type of routing schemes are called "memory-free" routing or "label-based" routing. As it is remarked in [9], every direct routing scheme can be transformed into a memory-free routing scheme by extending the size of each address so that it contains all the local data structures of the node.

This paper improves all these previous routing schemes for trees, including the one in [9].

Our contributions

1. We propose a direct shortest-path routing scheme for trees using $5 \log n$ bits for the addresses, $3 \log n$ bits of memory space, and $O(1)$ routing time[1]. This scheme is asymptotically optimal for all of the considered measures: memory space, header size, and routing time. Reducing the address range can be done, but up to an increase of the routing time. More precisely, we also propose a direct shortest-path routing scheme using $2.8 \log n$ bits for the addresses, $3 \log n$ bits of memory space, but involving polynomial time routing decisions.

   Attaching the address with the local data structure at each node, we thence show that there exists (1) a shortest-path memory-free routing scheme for trees with constant

[1] All the time complexities are computed in a word-RAM model where standard integer operations on words of $O(\log n)$ bits are performed in constant time, including logical bitwise operations, integer division, and integer multiplication. All the logarithms are in base 2.

routing time and $7 \log n$ bits of addresses, and (2) a shortest-path memory-free routing scheme for trees with a $4.8 \log n$ bits of addresses and polynomial routing time. All these results are stated in the designer-port model.

2. For the fixed-port model, we propose a direct shortest-path routing scheme using $O(\log^2 n / \log\log n)$ bits for the addresses, $O(\log^2 n / \log\log n)$ bits of memory space, and with constant routing time. We obtain a shortest-path memory-free routing scheme using $O(\log^2 n / \log\log n)$ bits with constant routing time. These results improve the best known result by a factor $O(\log\log n)$.

3. We also show lower bounds. In particular, we show that, in the designer-port, model any shortest-path routing scheme for arbitrary tree requires a memory space of at least $\alpha\sqrt{n} - o(\sqrt{n})$ bits if using $[1, n + o(\sqrt{n}/\log n)]$-range addresses. This improves and generalizes the lower bound of [3], and shows the optimality of the results therein. By comparing this lower bound with our first contribution, it also shows that a little variation of the address size, e.g., by an additive $O(\log n)$ bits factor, has a significant impact on the local memory space.

   For the fixed-port model, we prove a $n - o(n)$ bits lower bound on the memory space for shortest-path routing scheme on trees using $[1, n + o(n/\log n)]$-range addresses. In contrast with this bound, we construct a direct shortest-path routing scheme with $[1, n]$-range addresses, using at most $n + o(n)$ bits of memory space, and constant routing time. This demonstrates an optimal tradeoff between address-size and memory space.

2 A Routing Scheme with $O(\log n)$ bits of Address and Memory Space

Let T be any n-node tree, and let r be any node of T. From now, T is supposed to be rooted at r. For every node u, let us denote by T_u the subtree of T rooted at u induced by u and all its descendents. We denote by $w(u)$ the number of nodes of T_u, and we call it the *weight* of u. We define $\mathrm{id}(u)$ as the numbering of the nodes of T by consecutive integers in $[1, n]$ obtained by a DFS traversal of T performed under the following guideline: at each internal node, the DFS numbering visits first the largest subtree, then the second largest subtree, and so on. It yields a labeling in which $\mathrm{id}(r) = 1$, and, for every internal node u and every two of its children x, y, if $\mathrm{id}(x) \leqslant \mathrm{id}(y)$ then $w(x) \geqslant w(y)$.

2.1 Port and address assignment

If x is a child of y in T, then we define the *rank* of y, denoted by $\mathrm{rank}(x, y)$, as the number of children of x whose DFS labels are at most $\mathrm{id}(y)$. The labeling of the edge $\{u, v\}$ at u is denoted by $\mathrm{port}(u, v)$. We set $\mathrm{port}(r, v) = \mathrm{rank}(r, v)$ for any child v of the root r, and,

for any $u \neq r$,

$$\text{port}(u,v) \;=\; \begin{cases} 1 & \text{if } v \text{ is the parent of } u; \\ 1+\text{rank}(u,v) & \text{otherwise.} \end{cases}$$

The extra edge linking any node u to its local host is labeled 0. For every node $u \neq r$, we define $\text{path}(u)$ as the sequence of ranks encountered on the path from r to u. (We set $\text{path}(r) = ()$, the empty sequence.) More precisely, if the path from r to u is the sequence of nodes $r = u_0, u_1, \ldots, u_t = u$ then $\text{path}(u) = (\text{rank}(u_0,u_1), \ldots, \text{rank}(u_{t-1},u_t))$. We define the *clean-path* of u as the sequence $\text{cpath}(u)$ obtained from $\text{path}(u)$ by removing all the ranks that are equal to 1. We denote by $|\text{cpath}(u)|$ the length of the sequence $\text{cpath}(u)$, i.e., the number of ranks of $\text{path}(u)$ distinct from 1. In particular, $|\text{cpath}(u)| = 0$ if $\text{path}(u)$ consists of 1's only.

For every node u, the address $\ell(u)$ of u satisfies $\ell(u) = \langle \text{id}(u), \text{cpath}(u)\rangle$.

The data structure stored at node u is

$$\text{table}(u) \;=\; \langle \text{id}(u), w(u), w_1(u), |\text{cpath}(u)|\rangle$$

where $w_1(u)$ is the weight of the heaviest child of u, i.e., the weight of $w(z)$ such that $\text{rank}(u,z) = 1$. For a leaf u, we set $w_1(u) = 0$. The next lemma will be useful for computing the space needed for storing $\text{table}(u)$.

Lemma 1 *(1) If u is the parent of v, then* $\text{rank}(u,v) \leqslant w(u)/w(v)$.
(2) If $\text{cpath}(v) = (p_1, \ldots, p_k)$, *then* $\prod_{i=1}^{k} p_i \leqslant n/w(v)$, *and* $k \leqslant \log(n/w(v))$.

Proof.

(1) By definition, if $\text{rank}(u,v) = i$ then there are i children of u, say $c_1, \ldots, c_{i-1}, c_i = v$, such that $\text{id}(c_j) \leqslant \text{id}(v)$. For every such node c_j, $w(c_j) \geqslant w(v)$. Thus, $\sum_{j=1}^{i} w(c_j) \geqslant i \cdot w(v)$. Since $w(u) \geqslant \sum_{j=1}^{i} w(c_j)$, it follows that $w(u) \geqslant i \cdot w(v)$ that is $\text{rank}(u,v) \leqslant w(u)/w(v)$ as claimed.

(2) By construction, $p_i = \text{rank}(u_i, v_i)$ for some edge (u_i, v_i), u_i parent of v_i. By (1), $p_i \leqslant w(u_i)/w(v_i) \leqslant w(u_i)/w(u_{i+1})$ since $w(u_{i+1}) \leqslant w(v_i)$. It follows that $\prod_{i=1}^{k} p_i \leqslant w(u_1)/w(v_k)$. We have $w(u_1) \leqslant n$, and $v_k = v$. Since $p_i \geqslant 2$, $k \leqslant \log\left(\prod_{i=1}^{k} p_i\right) \leqslant \log(n/w(v))$. ∎

2.2 Routing algorithm

Our routing algorithm is direct, i.e., a message-header consists solely of the destination address of that message, and it will not be modified along its path from its source to its destination. Assume that a node x receives a message of header $h = \ell(y)$. The routing decision at x is described by a function $\textsc{Route1}(x,h) \in [0, \deg(x)]$ that returns the local label of the link on which the message has to be forwarded from x. In the following, let b denote the value 0 if $\text{id}(x) = 1$ (i.e., if $x = r$), and 1 otherwise.

$$\textsc{Route1}(x, \ell(y)) = \begin{cases} 0 & \text{if } \mathrm{id}(y) = \mathrm{id}(x); \\ 1 & \text{if } \mathrm{id}(y) < \mathrm{id}(x) \text{ or } \mathrm{id}(y) \geqslant \mathrm{id}(x) + w(x); \\ 1+b & \text{if } \mathrm{id}(x) < \mathrm{id}(y) \leqslant \mathrm{id}(x) + w_1(x); \\ p+b & \text{otherwise, where } p = (|\mathrm{cpath}(x)|+1)\text{-th element of } \mathrm{cpath}(y). \end{cases}$$

The next section shows that the routing algorithm is correct, and section 2.4 shows how to implement the function Route1.

2.3 Correctness

Lemma 2 *For every pair of nodes x, y, the routing algorithm described by* Route1 *routes any message from x to y along a shortest path.*

Proof.

If $\mathrm{id}(y) = \mathrm{id}(x)$, then the function returns the correct answer since the message is arrived at destination.

If $\mathrm{id}(y) < \mathrm{id}(x)$ or $\mathrm{id}(y) \geqslant \mathrm{id}(x) + w(x)$, then, by definition of the DFS numbering, y is not a descendent of x, and thus the message has to be transmitted to the parent of x. The routing function returns in this case the correct answer, i.e., port 1.

If $\mathrm{id}(x) < \mathrm{id}(y) \leqslant \mathrm{id}(x) + w_1(x)$ then y is a descendent of x. Since $\mathrm{id}(y) \leqslant \mathrm{id}(x) + w_1(x)$, the message has to be transmitted on the port of the heaviest child of x because the DFS numbering labels first the largest subtree of x. Therefore, the routing function correctly answers port $1 + b$ (i.e., 1 if $\mathrm{id}(x) = 1$, and 2 otherwise), because links are locally labeled in a decreasing order of the weights of the subtrees.

If $\mathrm{id}(x) + w_1(x) < \mathrm{id}(y) < \mathrm{id}(x) + w(x)$ then y is a descendent of x and the sequence $\mathrm{path}(x)$ is a prefix of $\mathrm{path}(y)$. Therefore, $\mathrm{cpath}(x)$ is also a prefix of $\mathrm{cpath}(x)$. The message has to be transmitted to a child of x, say z. Note that $\mathrm{path}(z) = (\mathrm{path}(x), \mathrm{rank}(x, z))$, and note also that $\mathrm{rank}(x, z) \neq 1$ because z is not the heaviest child of x since $\mathrm{id}(x) + w_1(x) < \mathrm{id}(y)$. Since $\mathrm{rank}(x, z) \neq 1$, and since $\mathrm{cpath}(x)$ is a prefix of $\mathrm{cpath}(y)$, we have that $\mathrm{rank}(x, z)$ is contained in the sequence $\mathrm{cpath}(y)$. The value $\mathrm{rank}(x, z)$ is precisely the first element of $\mathrm{cpath}(y)$ located after all the elements in common with $\mathrm{cpath}(x)$. Hence, it is the $(i+1)$-th element of $\mathrm{cpath}(y)$, where i is the number of elements of $\mathrm{cpath}(x)$. Thus, the routing function returns the correct answer in this case as well, completing the proof. ∎

2.4 Implementation

Lemma 3 *For every node u, its address $\ell(u)$ is of length less than $5 \log n + 1$ bits, and its local data structure* $\mathrm{table}(u)$ *is of length less than $3 \log n + \log \log n + 4$ bits.*

Proof. Recall that $\ell(u) = \langle \mathrm{id}(u), \mathrm{cpath}(u) \rangle$. We represent $\mathrm{id}(u)$ by a binary string of $\lceil \log n \rceil < (\log n) + 1$ bits. Now, let us give a compact representation of the sequence

$\text{cpath}(u) = (p_1, \ldots, p_k)$. Let l_i be the length of the binary decomposition of $q_i = p_i - 2$ ($q_i \geqslant 0$ since $p_i \geqslant 2$). Precisely, $l_i = \max\{\lceil \log(p_i - 1) \rceil, 1\}$. The sequence $\text{cpath}(u)$ is represented by two binary strings, say A and B, each of length $L = \sum_{i=1}^{k} l_i$, as follows: A is composed of the concatenation of the binary representation of the q_i's on l_i bits; and B is a bitmap that indicates the begining of each field representing a q_i in A (this is doable since $l_i \geqslant 1$).

Note that $p_i \geqslant 2$ implies $l_i < (\log p_i) + 1$. Hence, $L < \log\left(\prod_{i=1}^{k} p_i\right) + k$. By Lemma 1, $L < \log(n/w(u)) + \log(n/w(u)) \leqslant 2\log n$. In total, $\text{id}(u)$, A, and B can be represented by less than $5\log n + 1$ bits.

Finally, $\text{table}(u) = \langle \text{id}(u), w(u), w_1(u), k \rangle$, where $k = |\text{cpath}(u)|$, can be represented by a binary string of length at most $3\lceil \log n \rceil + \lceil \log k \rceil < 3\log n + \log\log n + 4$ bits (by Lemma 1, $k \leqslant \log n$). ∎

Actually, $\ell(u)$ can be represented in a more compact way. Indeed, the coding of $\text{cpath}(u)$ is not the best possible. To improve the coding we need the following lemma originally proved by Kalmár in 1930, and that can be also founded in [13].

Lemma 4 **[13]** *Let $Z(n)$ be the number of ordered integer sequences $p_1, p_2, \ldots$ such that $p_i \geqslant 2$ and $\prod_{i \geqslant 1} p_i \leqslant n$. Then, $Z(n) \sim n^{\rho}/\kappa$, with $\kappa = -\rho \cdot \zeta'(\rho)$ where ρ is the unique real solution of the equation $\zeta(\rho) = 2$ and ζ is the Riemann Zeta function. (We have $\rho \approx 1.7286$ and $\kappa \approx 3.1429$.)*

From this lemma, every clean-path can be coded using at most $\lceil \log Z(n) \rceil \leqslant \rho \log n < 1.8 \log n$ bits. Moreover, there exists a polynomial time algorithm that generates all such sequences[2]. Thus coding and decoding of $\text{cpath}(u)$ can be done in polynomial time.

Note that in order to execute $\textsc{Route1}(x, \ell(y))$ we need to extract rapidly each field of $\ell(y)$ and of $\text{table}(x)$. Given a binary string S of length L, let $\text{ones}_S(i)$ denote the function that returns the number of 1-entries in S up to (and including) position i, and let $\text{select}_S(j)$ denote the function that returns the position of the j-th 1-entry in S.

Lemma 5 **[7, 8]** *Let S be a binary string of length L function of n, such that $L \leqslant n$. The operations $\text{ones}_S(\cdot)$ and $\text{select}_S(\cdot)$ can be performed in $O(1)$ time with $o(L)$ pre-computed auxiliary bits, thus using a total of $L + o(L)$ space. Moreover, the auxiliary bits can be computed in $O(L)$ time.*

Theorem 1 *In the designer-port model, every n-node tree supports a direct shortest-path routing scheme with $5\log n + o(\log n)$ bits of address (resp., $2.8\log n$ bits), $3\log n + O(\log\log n)$ bits of memory space, and $O(1)$ (resp., $n^{O(1)}$) routing time. Moreover, the pre-processing time is $O(n \log n)$ (resp. $n^{O(1)}$).*

[2]For instance one can generate all the $\lfloor 4\log n \rfloor$ bit strings and test if it is a valid coding into the set of A, B strings of clean-paths.

Proof. The address of a node u is composed of three binary strings: $\mathrm{id}(u)$, and A and B to represents $\mathrm{cpath}(u)$. We enhance the address of u with $o(\log n)$ bits in order to support the operation $\mathrm{select}_B(\cdot)$ in constant time by Lemma 5. The data structure $\mathrm{table}(u)$ is not modified.

Let us show how to execute $\textsc{Route1}(x, \ell(y))$ in constant time, for any pair of nodes x, y. The three first conditions involve constant-time computations. (Note that extracting fields of fixed length and at a fixed position can easily be done by the use of binary masks and shifting.) Actually, only Condition 4 needs more attention due to the extraction of the $(i+1)$-th element of $\mathrm{cpath}(y)$. If $\mathrm{cpath}(y)$ is compacted in $\lceil \log Z(n) \rceil$ bits, that is by storing an index of the sequence $\mathrm{cpath}(y)$ in the set of all such sequences, it suffices to decode $\mathrm{cpath}(y)$ from its index, and to extract the $(i+1)$-th element. This takes $n^{O(1)}$ time. However, if $\mathrm{cpath}(y)$ is represented by two binary strings A and B, one can extract the $(i+1)$-th element in $O(1)$ time. Its value p is located in A between the position $j_1 = \mathrm{select}_B(i+1)$ and $j_2 = \mathrm{select}_B(i+2)$ (excluded). Both indices can be computed in constant time. Thanks to a mask $M = (2^{j_1+1} - 1) - (2^{j_2+1} - 1)$ applied on the string A, and at the price of a simple shift, one can extract p (actually, $p-2$) from A, and thus compute correctly $\textsc{Route1}(x, \ell(y))$.

Computing the weights requires $O(n)$ time, and sorting them at each node takes $O(n \log n)$. The time to perform the DFS numbering is then $O(n)$, and the time to compute all the clean-path sequences is then $O(n \log n)$ since every clean-path is on $O(\log n)$ bits (resp., $n^{O(1)}$ time if clean-paths are represented on $\lceil \log Z(n) \rceil$ bits). The time to compute all the addresses and data structures is again $O(n \log n)$, including the generation of auxiliary tables for each node. Therefore, the total pre-processing time is $O(n \log n)$ (resp. $n^{O(1)}$ if compacted). ∎

Concatenating the address and the local data structure of each node, we obtain (recall that $\ell(u)$ and $\mathrm{table}(u)$ have $\mathrm{id}(u)$ in common):

Corollary 1 *In the designer-port model, every n-node tree supports a memory-free shortest-path routing scheme with $7 \log n + o(\log n)$ bit for the addresses (resp. $4.8 \log n$ bits), and $O(1)$ (resp. $n^{O(1)}$) routing time. Moreover, the pre-processing time is $O(n \log n)$ (resp. $n^{O(1)}$) time.*

3 Routing Schemes for the Fixed-Port Model

As mentioned in the introduction, on one hand, the fixed-port model is potentially less efficient than the designer-port model in the sense that it may require the uses of larger data structures, but on the other hand it allows the combination of several routing schemes. This latter property makes the fixed-port model worth to be studied.

We reuse all the notations introduced in Section 2. In particular, for u parent of v, the fixed-port model assumes that $\mathrm{port}(u, v) \in [1, \deg(u)]$ is fixed, and cannot be modified. This has a significant impact on the size of the local data structures. Let us try to explain

why. Assume that the routing function deals with colored messages. Under the designer-port model, a compact routing scheme could consist to label locally the links with colors so that red messages are simply routed along the red link. Under the fixed-port model, this is not possible, and red messages may need to be routed along a green link. Note that one cannot simply recolor the red messages in green because there might be other nodes for which the routing of green messages must be performed through the blue link. Actually, the computation of "green" from "red" has an inherent cost, and requires additional memory bits. There are mostly two solutions for the storage of these bits. One consists to store them locally, e.g., node u stores a "correspondence table" of $O(d \log d)$ bits, where $d = \deg(u)$. Another consists to store them in the addresses of the nodes, e.g., all nodes w requiring the use of the link labeled $\mathrm{port}(u,v)$ to route messages from u to w have addresses containing additional bits for the computation of that label. The former solution is costly if d is large, e.g., it requires $\Omega(\log n)$ bits if $d = \Omega(\log n / \log\log n)$. The latter solution is costly if the path from u to w contains lots of nodes with large degree (such a path can contain up to $\Omega(\log n)$ nodes of degree $\Omega(\log n / \log\log n)$) because the address of w must then contain information about each of these nodes. Our strategy will mix these two solutions.

3.1 Address assignment

Let $f(n) = \Theta(\log n / \log\log n)$ (the reason for this setting will appear clear in forthcoming proofs). For any node u, we define the *light-path* of u as the sequence $\mathrm{lpath}(u)$ such that $\mathrm{lpath}(u) = ()$ if $u = r$, and if v is u's parent, then

$$\mathrm{lpath}(u) = \begin{cases} \mathrm{lpath}(v) & \text{if } \mathrm{rank}(v,u) \leqslant f(w(v)); \\ (\mathrm{lpath}(v), \mathrm{port}(v,u)) & \text{otherwise.} \end{cases}$$

Roughly speaking, $\mathrm{lpath}(u)$ contains only the large port numbers encountered while going from r to u. Let $\mathrm{cpath}(u) = (\mathrm{rank}(u_1,v_1), \ldots, \mathrm{rank}(u_k,v_k))$. We define $E(u)$ as the binary string of length k such that

$$E(u)[i] = \begin{cases} 0 & \text{if } \mathrm{rank}(u_i,v_i) \leqslant f(w(u_i)); \\ 1 & \text{otherwise.} \end{cases}$$

The address of u is similar to the one for the designer-port model, though it requires two extra fields, that is

$$\ell(u) \;=\; \langle \mathrm{id}(u), \mathrm{cpath}(u), E(u), \mathrm{lpath}(u) \rangle \;\; .$$

The local data structure of u is similar to the one for the designer-port model, though it requires an extra field $P(u)$, i.e.,

$$\mathrm{table}(u) \;=\; \langle \mathrm{id}(u), w(u), w_1(u), k, P(u) \rangle \;\; .$$

$P(u)$ is an array of $f(w(u)) + 1$ entries such that

$$P(u)[i] = \begin{cases} \mathrm{port}(u,x) & \text{if } i \in [1, f(w(u))] \text{ where } x \text{ satisfies } \mathrm{rank}(u,x) = i; \\ 0 & \text{if } i = 0 \text{ and } u = r; \\ \mathrm{port}(u,v) & \text{if } i = 0 \text{ and } u \neq r, \text{ where } v \text{ is the parent of } u. \end{cases}$$

3.2 Routing algorithm

Upon reception of a message of destination y, the routing decision at x is described by a function $\textsc{Route2}(x, \ell(y)) \in [0, \deg(x)]$ that returns the local label of the link on which the message has to be forwarded from x. As under the designer-port model, we make use of the $(|\text{cpath}(x) + 1|)$-th element p of $\text{cpath}(y)$ as follows. Let

$$q = \begin{cases} P(x)[p] & \text{if } E(y)[p] = 0; \\ j\text{-th element of } \text{lpath}(y) & \text{if } E(y)[p] = 1, \text{ where } j = \text{ones}_{E(y)}(p). \end{cases}$$

We then set

$$\textsc{Route2}(x, \ell(y)) = \begin{cases} 0 & \text{if } \text{id}(y) = \text{id}(x); \\ P(x)[0] & \text{if } \text{id}(y) < \text{id}(x) \text{ or } \text{id}(y) \geqslant \text{id}(x) + w(x); \\ P(x)[1] & \text{if } \text{id}(x) < \text{id}(y) \leqslant \text{id}(x) + w_1(x); \\ q & \text{otherwise.} \end{cases}$$

3.3 Correctness

Lemma 6 *For every pair of nodes x, y, the routing algorithm described by* Route2 *routes any message from x to y along a shortest path.*

Proof. The first three cases are trivial: x knows both the link leading to its parent ($P(x)[0]$) and the link leading to its heaviest child ($P(x)[1]$). Only the fourth case requires some attention. Similarly to Lemma 2,the $(|\text{cpath}(x) + 1|)$-th element of $\text{cpath}(y)$ is $p = \text{rank}(x, z)$ where (x, z) is the edge leading from x toward y. There are two cases:
Case 1. $E(y)[p] = 0$. Then $i = \text{rank}(x, z) \leqslant f(w(x))$, and thus $P(x)[i] = \text{port}(x, z)$ by definition of P. The function Route2 returns the correct answer in this case.
Case 2. $E(y)[p] = 1$. Then $i > f(w(x))$ and $\text{port}(x, z)$ is contained in $\text{lpath}(y)$. Note that the j-th port number along the clean-path from r to y is stored in $\text{lpath}(y)$ if and only if $E(y)[j]$ is set to 1. The number of 1-entries in $E(y)$ up to, and including, position p indicates how many port numbers are stored in $\text{lpath}(y)$. This allows to extract the j-th one by computing $j = \text{ones}_{E(y)}(p)$. The function Route2 returns the correct answer in this case as well, completing the proof. ∎

3.4 Implementation

The next lemma shows that both $\ell(u)$ and $\text{table}(u)$ can be encoded on slightly less bits than $\log^2 n$. Recall that $f(n) = \Theta(\log n / \log\log n)$.

Lemma 7 *For every node u, its address $\ell(u)$ and its local data structure* $\text{table}(u)$ *are both of size $O(\log^2 n / \log\log n)$ bits.*

Proof. Recall that $\ell(u) = \langle \mathrm{id}(u), \mathrm{cpath}(u), E(u), \mathrm{lpath}(u) \rangle$. The length of $E(u)$ is bounded by $|\mathrm{cpath}(u)|$. Thus the first three fields are all bounded by $O(\log n)$ bits. Assume that $\mathrm{lpath}(u) = (\mathrm{port}(u_1, v_1), \ldots, \mathrm{port}(u_t, v_t))$. We code $\mathrm{lpath}(u)$ via an array of t entries, where each entry is on $\lceil \log n \rceil$ bits. We have: $\mathrm{port}(u_i, v_i) \leqslant w(u_i)$, $w(u_{i+1}) \leqslant w(v_i)$, and $\mathrm{rank}(u_i, v_i) > f(w(u_i))$. By Lemma 1, we also have $\mathrm{rank}(u_i, v_i) \leqslant w(u_i)/w(v_i)$. Hence, $w(u_{i+1}) < w(u_i)/f(w(u_i))$. Let $g(n) = n/f(n)$. We have $w(u_{i+1}) < g(w(u_i))$. Thus, if we denote by $g^{(i)}(n)$ the i-th iteration of g, i.e., $g^{(0)}(n) = n$, and $g^{(i+1)}(n) = g(g^{(i)}(n))$, then

$$w(u_t) \;<\; g(w(u_{t-1})) \;<\; \cdots \;<\; g^{(t-1)}(w(u_1)) \;\leqslant\; g^{(t-1)}(n) \;.$$

Since $w(u_t) \geqslant 1$, it follows that $g^{(t-1)}(n) \geqslant 1$. Since $f(n) = \Theta(\log n / \log\log n)$, we have $g(n) = \Theta(n \log\log n / \log n)$, and therefore one can easily check that $t = O(\log n / \log\log n)$. Thus $\mathrm{lpath}(u)$ is coded by $O(t \log n) = O(\log^2 n / \log\log n)$ bits.

Recall that $\mathrm{table}(u) = \langle \mathrm{id}(u), w(u), w_1(u), |\mathrm{cpath}(u)|, P(u) \rangle$. The first four fields are on $O(\log n)$ bits each. $P(u)$ is composed of $f(w(u)) + 1$ entries, each one on $\lceil \log n \rceil$ bits. Hence $P(u)$ is of size at most $O(\log^2 n / \log\log n)$, completing the proof. ∎

Theorem 2 *In the fixed-port model, every n-node tree supports a direct shortest-path routing scheme with $O(\log^2 n / \log\log n)$ bits for both the addresses and the local memory spaces (thus a memory-free shortest-path routing scheme with $O(\log^2 n / \log\log n)$ bits addresses), and $O(1)$ routing time. Moreover, the pre-processing time is $O(n \log n)$.*

Proof. Given the implementation of Lemma 7, ROUTE2 can be computed in constant time, using the functions ones and select. The pre-precessing time does not increase, even though the size of the labels is slightly larger. Indeed, the pre-processing time for the operations $\mathrm{ones}_S(\cdot)$ and $\mathrm{select}_S(\cdot)$ are still performed on strings S of length $O(\log n)$, and hence it costs $O(\log n)$ time per node for its setup (cf. Lemma 5). ∎

3.5 A routing scheme with short range addresses

In this section, we focus on routing schemes using addresses of minimum size. In particular, we present a simple routing scheme with $[1, n]$-range addresses, and using $n + o(n)$ bits of local memory space. In Section 4 we will show that this scheme is optimal.

Theorem 3 *In the fixed-port model, every n-node tree supports a direct shortest-path routing scheme with $[1, n]$-range address, $n + o(n)$ bit memory space, and with constant routing time. The pre-processing time to setup this scheme is $O(n^2)$.*

Proof. Let T be an n-node tree such that, for every edge (u, v), $\mathrm{port}(u, v)$ has been fixed and cannot be modified. We consider T as rooted at an arbitrary node, say r. We define the address $\ell(u)$ of u by labeling the nodes with integers in $[1, n]$ using a DFS traversal of

T performed as follows: $\ell(r) = 1$, and, at each internal node u, for any two children v_1 and v_2 of u, if $\mathrm{port}(u, v_2) > \mathrm{port}(u, v_1)$, then v_1 is visited before v_2.

For any node u, we also define the value $m(u)$ as the largest value of $\ell(w)$ among all descendents w of u. We finally define $B(u)$ as the binary array of n bits such that $B(u)[i] = 1$ if and only if u has a child v such that $\ell(v) = i$. The local data structure of a node u is $\mathrm{table}(u) = \langle \ell(u), m(u), B(u), p(u) \rangle$ where $p(u) = \mathrm{port}(u, v)$ with v the parent of u ($p(u) = 0$ if $u = r$).

The routing algorithm ROUTE3 is the following:

$$\textsc{Route3}(x, \ell(y)) = \begin{cases} 0 & \text{if } \ell(y) = \ell(x); \\ p(x) & \text{if } \ell(y) > m(x) \text{ or } \ell(y) < \ell(x); \\ q + b & \text{otherwise, where } q = \mathrm{ones}_{B(x)}(\ell(y)) \text{ and } b = \begin{cases} 0 & \text{if } q \geqslant p(x) \\ 1 & \text{otherwise.} \end{cases} \end{cases}$$

The correctness of the algorithm is trivial in the two first cases. For the third case, note first that if $v_1, \ldots, v_t$ are the children of x such that $\mathrm{port}(x, v_1) < \cdots < \mathrm{port}(x, v_t)$, and if y belongs to some T_{v_i}, then the message has to be transmitted to v_i and the corresponding port number is i if $x = r$ or if $i < p(u)$, and $i + 1$ if $x \neq r$ or $i \geqslant p(u)$. The test "y belongs to T_{v_i}?" can be performed by testing whether $\ell(v_i) \leqslant \ell(y) < \ell(v_{i+1})$, that is by computing the number of 1-entries in $B(x)$ up to (and including) position $\ell(y)$. Thus ROUTE3 returns the correct answer in the third case too. (Note that if $x = r$, the condition $q \geqslant p(x)$ is always satisfied because $p(r) = 0$ and $q \geqslant 0$, thus $b = 0$.)

The addresses are in the range $[1, n]$, and the local data structure is of size $n + o(n)$, including the data structure for the computation of $\mathrm{ones}_{B(x)}(\cdot)$ in constant time. The routing time is therefore $O(1)$, and the pre-processing time is $O(n)$ per node (the time to compute $B(x)$ and required to compute the $o(n)$ bits auxiliary bits for the ones operation, cf. Lemma 5). ∎

4 Lower Bounds for Short Range Addresses

Computing lower bounds often requires the use of *routing strategy*. A routing strategy $\mathcal{R}$ is simply a function that returns, for every graph G of a given family $\mathcal{F}$, a routing scheme $R = \mathcal{R}(G)$ for G. We denote by $\textsc{Mem}(R, x)$ the set of all the data structures used by R in node x of G.

Let $\mathcal{M}$ be any set, and let k be an integer. A *k-protocol* w.r.t. $\mathcal{R}$ and $\mathcal{M}$ is a pair of functions (s, m) such that, for any $G \in \mathcal{F}$, and for any $x \in V(G)$, we have

$$(1)\ \ s(\mathcal{R}(G), G, x) \in \{0,1\}^k \ \text{ and } (2)\ \ m(s(\mathcal{R}(G), G, x), \textsc{Mem}(\mathcal{R}(G), x)) \in \mathcal{M}.$$

A k-protocol (s, m) is said *complete* if for any $D \in \mathcal{M}$, there exists $G \in \mathcal{F}$, and $x \in V(G)$ such that

$$m(s(\mathcal{R}(G), G, x), \textsc{Mem}(\mathcal{R}(G), x)) = D.$$

We can prove the following:

Lemma 8 *Given a set $\mathcal{M}$, and a routing strategy $\mathcal{R}$ on $\mathcal{F}$, if there exists a complete k-protocol, then there exists $G \in \mathcal{F}$ and $x \in V(G)$ such that the size of* $\textsc{Mem}(\mathcal{R}(G), x)$ *is at least* $\log|\mathcal{M}| - k$ *bits.*

Proof. Let (s, m) be a complete k-protocol. Each element $D \in \mathcal{M}$ can be coded as a pair $\langle S, M\rangle$, where $S = s(\mathcal{R}(G), G, x)$, $M = \textsc{Mem}(\mathcal{R}(G), x)$, $G \in \mathcal{F}$ and $x \in V(G)$ are such that $m(S, M) = D$. By construction, given its code $\langle S, M\rangle$, we can obtain D by applying the function m. Let L be the size of $\textsc{Mem}(\mathcal{R}(G), x)$. The size of the coding of $D \in \mathcal{M}$ described above is at most $k + L$. If $k + L < \log|\mathcal{M}|$, then two distinct elements of $\mathcal{M}$ have the same encoding $\langle S, M\rangle$: a contradiction. Therefore, $L \geqslant \log|\mathcal{M}| - k$. ∎

By comparison with the routing scheme in Theorem 3 (fixed-port model) and the routing scheme in [4] (designer-port model), the two following lower bounds are optimal.

Theorem 4
In the fixed-port model, every shortest path routing strategy on n-node trees with $[1, n + o(n/\log n)]$-range addresses requires at least $n - o(n)$ bits for some node of some tree.
In the designer-port model, every shortest path routing strategy on n-node trees with $[1, n + o(\sqrt{n}/\log n)]$-range addresses requires at least $\alpha\sqrt{n} - o(\sqrt{n})$ bits for some node of some tree.

Proof. Let us denote by $\mathcal{T}_1$ the family of unlabeled n-node trees, and by $\mathcal{T}_2$ the family of n-node trees such that every out-going edge of every node is labeled by a unique integer between 1 and the degree of that node. Moreover, let $\mathcal{M}_1$ be the set of sequences of non-null integers $n_1, n_2, \ldots$ such that $\sum_{i\geqslant 1} n_i = n - 1$. Let $\mathcal{M}_2$ be the set of non-decreasing sequences of non-null integers $n_1, n_2, \ldots$ such that $\sum_{i\geqslant 1} n_i = n-1$. Note that $|\mathcal{M}_1| = 2^{n-1}$ and that $|\mathcal{M}_2| = 2^{\alpha\sqrt{n}+o(\sqrt{n})}$ with $\alpha = (\pi\sqrt{2/3})/\ln 2 \approx 3.7$. (The latter formula comes from the fact that the number of ways to write n in non-null summands is asymptotically $\frac{1}{4n\sqrt{3}}\, e^{\pi\sqrt{2n/3}}$, the well-known Harder-Ramanujan's formula, cf. [6, Equation (4.2.7) page 44].)

Proving the two lower bounds requiring very similar technique, we give their proofs simultaneously.

Let $\mathcal{T} \in \{\mathcal{T}_1, \mathcal{T}_2\}$. For an arbitrary shortest-path routing strategy $\mathcal{R}$ on $\mathcal{T}$ using $[1, n+r]$-range addresses (we take $r = o(\sqrt{n}/\log n)$ in the fixed-port model, and $r = o(n/\log n)$ in the designer-port model), we define a complete k-protocol (s, m) as follows. For every $T \in \mathcal{T}$ and every $x \in V(T)$, we define $s(\mathcal{R}(T), T, x) = S$ where S the set of all the addresses assigned by $\mathcal{R}(T)$ distinct from the address of x. We define $m(S, \textsc{Mem}(\mathcal{R}(T), x)) = (n_1, \ldots, n_d)$ where n_i is the number of nodes such that the routing function in x, applied to all the addresses in S, returns the port number i. (For the designer-port model the n_i's are sorted so that $n_1 \leqslant \cdots \leqslant n_d$).

By construction, the sequence returned by m is in $\mathcal{M}_1$ in the fixed-port model, and in $\mathcal{M}_2$ in the designer-port model. Moreover, the k-protocol (s, m) is complete because, for every sequence $(n_1, \ldots, n_d)$, there exists a tree $T \in \mathcal{T}$, and a node x of degree d in T such that, for every child u of x:

- In the fixed-port model: $\mathrm{port}(x,u) = i$ implies $w(u) = n_i$;
- In the designer-port model: $w(u) = n_i$.

By Lemma 8, for any routing strategy $\mathcal{R}$ with $[1, n+r]$-range addresses, there is a tree $T_0 \in \mathcal{T}$ and a node $x_0 \in V(T_0)$, such that the local memory space of x_0 is at least $\log|\mathcal{M}| - k$. It remains to show that $k = o(n)$ in the fixed-port model, and $k = o(\sqrt{n})$ in the designer-port model.

There are at most $\binom{n+r}{n-1}$ possible sets of $n-1$ addresses in $[1, n+r]$. Thus $k \leqslant \log\left\lceil \binom{n+r}{n-1} \right\rceil$. Since $n + r \leqslant 2(n-1)$, we have

$$\binom{n+r}{n-1} = \binom{n+r}{r+1} \;\leqslant\; (2(n-1))^{r+1} \quad .$$

Therefore, $k \leqslant r\log n + O(r + \log n)$, that is $k = o(n)$ if $r = o(n/\log n)$, and $k = o(\sqrt{n})$ if $r = o(\sqrt{n}/\log n)$, which completes the proof. ∎

5 Conclusion

We have shown that n-node trees support routing schemes with message headers, node addresses, and local memory space of size $O(\log n)$ bits, and such that every local routing decision is taken in constant time. Beside this result, we have shown that little variations of the address size (by constant multiplicative factors, or logarithmic additive factors) can have a tremendous impact on the local memory space. In other words, the multiplicative constants hidden by the big-O notations play a significant role on the address-range vs. memory-space trade-off for routing. We can actually show similar results for space vs. time trade-offs. E.g., for some trees, small headers (i.e., in $[1, n + o(\sqrt{n}/\log n)]$) and compact data structures, (i.e., of size $2\log n$) yield situations for which routing along shortest-path is doable but requires exponential times. It is actually even possible to show that the problem of whether there exists a direct shortest-path routing scheme with $[1, n + o(\sqrt{n}/\log n)]$-range addresses, and using local data structures of optimal size up to a multiplicative factor of $\sqrt{n}/\log n$ is undecidable for any enumerable family of graphs containing n-node trees. All these results will be presented in the full version of the paper. As a last concluding remark, we conjecture that Theorem 2 is optimal, i.e., $\Omega(\log^2 n/\log\log n)$ bits for the addresses and the local data structures is a lower bound under the fixed-port model.

Acknowledgment: The authors are thankful to Michel Balazard and Michel Mendès France for showing them reference [13].

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] M. J. Hall, *Combinatorial Theory (second edition)*, Wiley-Interscience Publication, 1986.

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

[8] 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.

[9] D. Peleg, *Proximity-preserving labeling schemes and their applications*, in 25^{th} International Workshop, Graph - Theoretic Concepts in Computer Science (WG), vol. 1665 of Lecture Notes in Computer Science, Springer, June 1999, pp. 30–41.

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

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

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

[13] R. Warlimont, *Factorisatio numerorum with constraints*, Journal of Number Theory, 45 (1993), pp. 186–199.