Compact Name-Independent Routing with Minimum Stretch

Ittai Abraham
School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel
ittaia@cs.huji.ac.il

Cyril Gavoille*
Laboratoire Bordelais de Recherche en Informatique
University of Bordeaux
Bordeaux, France
gavoille@labri.fr

Dahlia Malkhi †
School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel
dalia@cs.huji.ac.il

Noam Nisan
School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel
noam@cs.huji.ac.il

Mikkel Thorup
AT&T Labs - Research
Shannon Laboratory
Florham Park, NJ 07932, USA
mthorup@research.att.com

ABSTRACT

Given a weighted undirected network with arbitrary node names, we present a compact routing scheme, using a $\widetilde{O}(\sqrt{n})$ space routing table at each node, and routing along paths of stretch 3, that is, at most thrice as long as the shortest paths. This is optimal in a very strong sense. It is known that no compact routing using $o(n)$ space per node can route with stretch below 3. Also, it is known that any stretch below 5 requires $\Omega(\sqrt{n})$ space per node.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed networks*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems, Graph labeling*.

General Terms

Algorithms, Theory.

Keywords

Compact Routing.

*Supported by the project "PairAPair" of the ACI Masses de Données.

†This work was supported in part by the Israeli Science Foundation (246/02) and by EC Evergrow project.

SPAA'04, June 27–30, 2004, Barcelona, Spain.

1. INTRODUCTION

Routing is one of the fundamental problems of distributed systems. Consider that we are given a set V of n nodes with names chosen as an arbitrary permutation on $\{1,\ldots,n\}$. Also given is a weighted, undirected graph $G=(V,E,\omega)$ indicating the connectivity between nodes and the edge cost ω. A *routing scheme* is a distributed algorithm that allows any node u to route messages to any other node $v\in V$. In order to reduce memory overhead and incur costs that are proportional to the actual distances between interacting parties, there are two parameters routing schemes aim to minimize:

1. *Stretch*: the ratio between the cost of the path taken by the routing protocol and a minimum cost path from source to destination.

2. *Memory*: the number of bits stored in each node.

The trivial solution to routing with optimal stretch 1 is to store the complete routing table of the all pairs shortest path algorithm. This solution requires each node to store $n\log n$ bits and thus does not scale well as the number of nodes in the system increases. The challenge is in minimizing the stretch for *compact* solutions, in which each node stores $o(n)$ bits. The known lower bound in effect is that of Gavoille and Gengler [10], indicating at least stretch 3 when each node has memory $o(n)$. For comprehensive surveys on compact routing and compact network data structures, see [9, 11].

The minimal stretch compact routing problem has two basic variants: *labeled routing* and *name independent routing*. Awerbuch et al. were the first to distinguish in [2] between solutions that allow/disallow the designer to choose labels for nodes as part of the solution. The variant that allows the designer to name nodes with arbitrary labels is called *labeled routing*. In this model, the node's label may contain valuable topology dependent information useful for routing, usually with poly-log size labels. This tends to make the design of solutions easier. The variant that does not allow labeling of nodes in this way is called *name independent routing*. In this variant node names may be chosen arbitrarily by an adversary. This makes routing generally harder: Intuitively, the routing algorithm must first discover information about the location of the target, and only then route to it.

Indeed, optimal stretch 3 compact schemes for labeled routing are already known. The first stretch 3 scheme was given by Cowen [6] with $\widetilde{O}(n^{2/3})$ memory[1]. Later, Thorup and Zwick [17, 18] improved the memory bound to only $\widetilde{O}(\sqrt{n})$ bits. They also gave an elegant generalization of their scheme, achieving stretch $4k-5$ (and even $2k-1$ with handshaking) using only $\widetilde{O}(n^{1/k})$ bits. Additionally, there exist various labeled routing schemes suitable only for certain restricted forms of graphs. For example, routing in a tree is explored, e.g., in [7, 18], achieving optimal routing. This routing requires $O(\log^2 n/\log\log n)$ bits for local tables and for headers, and this is tight [8].

As for name independent routing, the situation is quite different. Initial results in [3] provide non-compact name independent routing with $\widetilde{O}(n^{3/2})$ total memory. Awerbuch and Peleg [4] were the first to show that constant-stretch is possible to achieve with $o(n)$ memory per node, albeit with a large constant. Recently, Arias et al. significantly reduced the stretch factor in [1], providing stretch 5 with $\widetilde{O}(\sqrt{n})$ memory per node. However, these results leave a gap between the known lower bound of stretch 3.

1.1 Our results

We present the first optimal compact name independent routing scheme for arbitrary undirected graphs. The scheme has stretch 3, and memory complexity $\widetilde{O}(\sqrt{n})$ per node. Given the graph and the node names, we can construct the routing information deterministically in polynomial time. Moreover, when routing along our stretch 3 paths, each routing decision is performed in constant time.

Besides improving Arias et al. [1] stretch from 5 to 3, our results answer affirmatively the challenge of optimal name independent routing that was open since the initial statement of the problem in 1989 [3]. Surprisingly, our results show that allowing the designer to label the nodes does **not** improve the stretch factor compared to the task when node labels are predetermined by an adversary.

We note that our solution does not contain any strikingly new technique. Rather our new scheme is a non-trivial combination of simple standard techniques.

2. PRELIMINARIES

Consider a set V of n nodes wishing to participate in a distributed routing scheme. We assume the nodes are labeled with an arbitrary permutation of the integers $\{1,\ldots,n\}$.

We assume a graph $G=(V,E,\omega)$ with nonnegative edge cost ω. For $u,v\in V$, let $d(u,v)$ denote the cost of a minimum path from u to v in G, were the cost of a path is the sum of the weights of along its edges.

Each node has, for each outgoing edge, a unique name from the set of integers $\{1,\ldots,n\}$. We assume the fixed-port model [7]. In this model the name of the outgoing edge is fixed *before* the adversary chooses the permutation of node labels. Thus the name of the outgoing edge has no connection to the label of the node on the other side of the edge.

When a node wants to send a message, we assume that initially the sender only knows the name of the destination node. This destination is written in the header of the message. We require *writable packet headers*, namely, we allow the routing algorithm to write a reasonable amount of information into the headers of messages as they are routed. In our case, we use $O(\log^2 n/\log\log n)$ bit headers.

We note that our use of headers aim at useful tradeoffs between current techniques used in the real world: source directed routing, where the source puts the whole path to the destination in the header, and routing based on routing tables each router knows how to forward packets to any destination. For source directed routing, the header may be very large, and for the other routing, the routing tables may become huge. In either case, we have problems with scaling. Our point here is that a small amount of information in the header can dramatically reduce the amount of information needed at the routers.

It is no coincidence that our scheme and indeed all previous name independent schemes use writable packet headers. A scheme that does not re-write packet headers must be loop free and thus must have stretch 1 on any tree. Clearly in a tree that is a star the center would have to code a permutation using $\Omega(n\log n)$ bits on the average.

Lemma 2.1. *There do not exist loop free name independent routing schemes with $o(n)$ bits for each node on every graph.*

As for lower bounds for compact routing, note that for the related problem of labeled routing, the work of [10] shows that any stretch <3 scheme must use a total of $\Omega(n^2)$ bits. Thus it cannot be the case that all nodes use $o(n)$ bits. This bound clearly holds also for the name independent model.

Actually, a slightly stronger memory bound of $\Omega(n^2\log n)$ bits for stretch <3 can be proven for the name independent model. This is derived by examining the complete bipartite graph $K_{n/2,n/2}$ with uniform weights (likewise, the metric space it induces). For stretch <3, each node must route optimally to its distance one neighbors. By counting all the permutations on names it is clear that each node must use $\Omega(n\log n)$ bits.

Lemma 2.2. *Any name independent routing scheme with $o(n\log n)$ bits per node must have stretch at least* 3.

3. THE STRETCH 3 SCHEME

In Sections 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, we will first present some simple ingredients for our optimal stretch 3 scheme. Then, in Section 3.8, we will combine them in our optimal solution. Finally, in Section 3.9, we prove that the scheme has the optimal stretch of 3.

3.1 Vicinity balls

For every integer $\kappa\geq 1$, and for a node $u\in V$, let the *vicinity* of u, denoted by $B_\kappa(u)$, be the set of the κ closest nodes to u, breaking ties by lexicographical order of node names. Vicinities satisfy the following Monotonicity Property:

Property 3.1. **[3]** *For any $\kappa\geq 1$, if $v\in B_\kappa(u)$, and if w is on a minimum cost path from u to v, then $v\in B_\kappa(w)$.*

[1]The notation $\widetilde{O}()$ indicates complexity similar to $O()$ up to poly-logarithmic factors.

Hereafter, the size of the vicinities is set to $\kappa = \lceil 4\alpha\sqrt{n}\log n\rceil$, where $\alpha > 2$ is some constant fixed in Section 4 and denote simply by $B(u) = B_\kappa(u)$. Let $b(u)$ denote the radius of $B(u)$, $b(u) = \max_{w\in B(u)} d(u,w)$.

As in previous compact routing schemes (see, e.g., [2, 6]), each node u will know its vicinity $B(u)$. We assume that u has a standard dictionary over the names in $B(u)$ so that in constant time it can check membership and look up associated information.

3.2 Coloring

Our construction uses a partition of nodes into sets $C_1,\ldots,C_{\sqrt{n}}$, called *color-sets*, with the following two properties:

Property 3.2.

1. *Every color-set has at most $2\sqrt{n}$ nodes.*

2. *Every node has in its vicinity at least one node from every color-set.*

From here on, if node $u \in C_i$ we say that it has "color i", and denote $c(u) = i$. Property 3.2 clearly holds w.h.p. if every node independently chooses a random color. Constructing a polynomial-time coloring satisfying Property 3.2 is discussed in Section 4.

3.3 Hashing names to colors

We shall assume a mapping h from node names to colors which is balanced in the sense that at most $O(\sqrt{n}\log n)$ names map to the same color. Each node u should be able to compute $h(w)$ for any destination w. If the names were a permutation of $\{1,\ldots,n\}$, we could just extract $\frac{1}{2}\log n$ bits from the name, but we want to deal with arbitrary names such as IP addresses. Arias et al. [1] suggest to use a $(\log n)$-universal hash function for a similar purpose. In Section 5 we will present a function h that can be computed in constant time.

3.4 Stretch 3 for metric spaces

To illustrate the use of these first three ingredients, we here observe a very simple stretch 3 scheme with $\widetilde{O}(\sqrt{n}\,)$ bits per node for any metric space, i.e., for a distributed network in which the cost of communication between nodes is governed by a distance function.

Every node u stores the following:

1. The names of all the nodes in $B(u)$.

2. The names of all the nodes v such that $c(u) = h(v)$.

Routing from u to v is done in the following manner:

1. If $v \in B(u)$ or $c(u) = h(v)$ then u routes directly to v with stretch 1.

2. Otherwise, u forwards the packet to $w \in B(u)$ such that $c(w) = h(v)$. Then from w the packet goes directly to v. The stretch is at most 3 since $d(u,w) + d(w,v) \le d(u,v) + 2d(u,v)$.

Note that in a general graph the main difficulty is in implementing the path from w to v.

3.5 Routing on trees

We make use of the following result concerning topology-dependent labeled routing:

Lemma 3.3. **[7, 18]** *There is a routing scheme for any tree T with n nodes that routes optimally between every pair of nodes. The storage per node in T and the header size are $O(\log^2 n/\log\log n)$ bits. Given the information of a node and the label of the destination, routing decisions take constant time.*

For a tree T containing a node v, we let $\mu(T,v)$ denote the routing information of node v from Lemma 3.3 and $\lambda(T,v)$ denote the destination label of v in T.

We shall apply Lemma 3.3 to several trees in the graph. Each node will participate in $\widetilde{O}(\sqrt{n}\,)$ trees, so its total tree routing information will be of size $\widetilde{O}(\sqrt{n}\,)$.

3.6 Landmarks

Borrowing from [3, 6], we designate one color to be special, e.g., we set color 1 as the special color and call its color *red*. We use the red nodes as routing *landmarks*. Let L denote the set of red nodes. We have $|L| \le 2\sqrt{n}$. We have already shown how to assign colors such that for every $v \in V$, the vicinity $B(v)$ contains one red node from L. For a node $v \in V$, let ℓ_v denote an arbitrarily red node in $B(v)$. For any red node $\ell \in L$, denote by $T(\ell)$ the single-source minimum cost tree rooted at ℓ. Note that there are only $\widetilde{O}(\sqrt{n}\,)$ red trees.

3.7 Partial shortest path trees

For any node u let $T(u)$ denote the single-source minimum cost tree rooted at u. In a partial shortest path tree, every node v maintains $\mu(T(u),v)$ if and only if $u \in B(v)$. Notice that the set of nodes that maintain $\mu(T(u),\cdot)$ is a subtree of $T(u)$ that contains u.

Lemma 3.4. *If $x \in B(y)$ then given the label $\lambda(T(x),y)$, node x can route to node y along a minimum cost path.*

Proof. By Property 3.1 for any node w on the minimum cost path of $T(x)$ between x and y we have $x \in B(w)$. Thus every node w on this path maintains $\mu(T(x),w)$. $\square$

3.8 The stretch 3 scheme

Every node u stores the following:

1. The names of all the nodes in the vicinity $B(u)$ and what link to use to reach them.

2. Routing information $\mu(T(\ell),u)$ of the tree $T(\ell)$ for every red node $\ell \in L$.

3. Routing information $\mu(T(x),u)$ of the tree $T(x)$ for every node $x \in B(u)$.

4. The names of all the nodes v such that $c(u) = h(v)$. For every node v such that $c(u) = h(v)$, store one of the following two options that produces the minimum cost path out of the two:

   (a) Store the labels $\langle\,\lambda(T(\ell_v),\ell_v),\ \lambda(T(\ell_v),v)\,\rangle$. The routing path in this case would be from u to $\ell_v \in B(v)$ using $\lambda(T(\ell_v),\ell_v)$ on the tree $T(\ell_v)$, and from ℓ_v to v using $\lambda(T(\ell_v),v)$ on the tree $T(\ell_v)$.

(b) Let $P(u,w,v)$ be a path from u to v composed of a minimum cost path from u to w, and of a minimum cost path from w to v with the following properties: $u \in B(w)$, and there exists an edge (x,y) along the minimum path from w to v such that $x \in B(w)$ and $y \in B(v)$. Among all these paths choose the lowest cost path $P(u,w,v)$ and store the labels $\langle\, \lambda(T(u),w),\, x,\, (x \rightarrow y),\, \lambda(T(y),v)\,\rangle$.

The routing path in this case would be from u to w on $T(u)$ using $\lambda(T(u),w)$. This part is possible by Lemma 3.4 on $u \in B(w)$. Then from w to y since $x \in B(w)$ and the port number $(x \rightarrow y)$ is stored. Finally from y to v on $T(y)$ using $\lambda(T(y),v)$. This part is possible by Lemma 3.4 on $y \in B(v)$.

Routing from u to v is done in the following manner:

1. If $v \in B(u)$ or $v \in L$ (v is a red node) or $c(u) = h(v)$ then u routes to v using its own information.

2. Otherwise, u forwards the packet to $w \in B(u)$ such that $c(w) = h(v)$. Then from w the packet goes to v using w's routing information.

3.9 Analysis

Theorem 3.5. *Let $s,t \in V$ be any two nodes. The route of the above scheme from s to t has stretch at most* 3.

Proof. There are three cases to consider:

1. If $t \in B(s)$ or $t \in L$ then s routes on a minimum cost path directly to t.

Otherwise, denote $d = d(s,t)$, let z be a node such that $z \in B(s)$ and $c(z) = h(t)$. For the case $c(s) = h(t)$, we set $z = s$. Let $p(z,t)$ be the cost of the path chosen by z as the lowest cost path from z to t among options 4a and 4b.

2. If on every minimum cost path from s to t there is a node y such that $y \notin B(s)$ and $y \notin B(t)$ then $b(s) + b(t) \leq d(s,t)$.

   By examining option 4a the cost $d(s,z)+p(z,t)$ of the path taken by our routing scheme is bounded by the cost of the path $s \leadsto z \leadsto \ell_t \leadsto t$. Thus $d(s,z) + p(z,t) \leq d(s,z) + d(z,\ell_t) + d(\ell_t,t) \leq b(s) + [b(s) + d + b(t)] + b(t) \leq 3d$.

3. If there exists a minimum cost path, in which every node is in $B(s) \cup B(t)$ then let (x,y) be an edge of this path such that $x \in B(s)$ and $y \in B(t)$.

   By examining the best choice in option 4b the cost $d(s,z)+p(z,t)$ of the path taken by our routing scheme is bounded by the cost of the path $s \leadsto z \leadsto s \leadsto x \rightarrow y \leadsto t$. Thus $d(s,z) + p(z,t) \leq d(s,z) + d(z,s) + d(s,t) \leq b(s) + b(s) + d \leq 3d$.

□

4. ON POLYNOMIAL TIME COLORING

In this section, we discuss how to "derandomize" the coloring discussed in Section 3.2. We shortly describe how the coloring of vertices can be done deterministically in polynomial time. Let us first consider our coloring problem in a slightly more abstract setting.

Given are m subsets of $\{1,\ldots,n\}$, $B_1,\ldots,B_m$, where $|B_i| \geq \alpha k \log n$, for all i where k is a parameter and α is some constant large enough. Our task is to color the n items with k colors, $c : \{1,\ldots,n\} \rightarrow \{1,\ldots,k\}$, such that the following properties hold: (a) Each color appears at most $(\alpha n \log n)/k$ times; (b) Each color appears in each set B_i. (In our application the n items are the vertices, $m = n$, the m sets are the vicinities of the vertices, and $k = \sqrt{n}$.)

Clearly a random coloring will do the trick, as long as m is polynomial in n. Very coarse calculations reveal that the probability that any fixed condition of type (a) or type (b) is not satisfied is bounded by $n^{-\alpha}$, for any value of k. Thus the probability that some condition is not satisfied is bounded by $(mk+k)n^{-\alpha}$.

How can we derandomize this construction and obtain an explicit construction? While $O(\sqrt{n}\log n)$-wise independence suffices for the argument above, this still does not provide a deterministic construction. It turns out that this can be done using the pseudo-random generators for space bounded computation of [14]. Each of the conditions can be computed in Logspace and in [13] it is shown how the seed of the pseudo-random generator can be incrementally chosen as to pass a Logspace test.

Specifically, we can look at the output string of the generator as being $y_1,\ldots,y_n$, where each y_i is of length $\log k$ bits and denotes the color of item i. Now, a Logspace machine with a one-way access to this stream, can verify by simple counting, for any fixed color j, that color j appears at most $(\alpha n \log n)/k$ times. Similarly, such a machine can verify, for any fixed B_i and any fixed color j, that B_i includes an element of color j. It follows that the output of the generator will also almost surely pass each one of these tests.

Since the generator of [14] accepts $O(\log^2 n)$ bits as input, this is still not a complete derandomization. However, in [13] it is shown how the input bits of the generator can be incrementally chosen in polynomial time such that the output passes a given Logspace test. As described in detail in [15], the same algorithm with the same argument actually allows choosing the input bits such that the output passes a given *polynomial size family* of Logspace tests. As mentioned above, this is exactly the case here.

5. ON HASHING IN CONSTANT TIME

In this section, we implement a constant time hashing function from names to colors assumed in Section 3.3

Suppose that the node names are taken from an arbitrary space, e.g., IP addresses. We will now find a more efficient hash function h mapping the n vertex names into the colors $0,\ldots,\sqrt{n}$. The mapping should be balanced in the sense that only $O(\sqrt{n}\log n)$ names map to any color.

The solution by Arias et al. [1] was to let h be a $(\log n)$-universal hash function, but with current implementations via degree-$(\log n)$ polynomials, the evaluation of this function takes more than constant time. We suggest an alternative implementation of h which takes constant time to evaluate. Consequently, each routing decision will take constant time with our scheme.

The representation of our hash function will take $O(\sqrt{n})$ space, but we can still store such a representation with each node without violating our space bounds.

First we use a standard universal hash function h_0 mapping names into $[n^{2.5}]$ in constant time. With high probability this mapping is collision free, that is, no two names map to the same reduced name. We can check that h_0 is collision free, and if not, try another h_0.

Set $p = (\log n)/2$. We are now dealing with reduced names of $5p$ bits, and we want to get down to colors of p bits. We will use an idea of Tarjan and Yao [16]. For $i = 1, \ldots, 4$, let T_i be a random table mapping p bits into $(5 - i)p$ bits. Note that each table has $2^p = \sqrt{n}$ entries. We then hash a $(5p)$-bit reduced name x as follows. For $i = 1, \ldots, 4$, let y be the p least significant bits of x, and set $x = T_i[y] \oplus (x \gg p)$. At the end, x has only $p = (\log n)/2$ bits which we return as the color.

The above computation of colors from reduced names takes constant time, and it is straightforward to show that we expect a maximum of $O(\sqrt{n} \log n)$ reduced names to map to the same color. Since the initial mapping to reduced names took constant time and was collision free, we conclude that the overall mapping takes constant time and maps only $O(\sqrt{n} \log n)$ names to each color.

The above construction use the same ingredients as are used for the deterministic dictionaries of Hagerup et al. [12]. Using the techniques from [12], we can derandomize our construction to run in $O(n \log n)$ time, yielding the desired balanced constant-time mapping from names to colors.

6. ACKNOWLEDGEMENTS

We would like to thank the anonymous referee for helpful comments.

7. REFERENCES

[1] M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka. Compact routing with name independence. In *Proceedings of the 15th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 184–192, 2003.

[2] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *Proceedings of the 21st annual ACM Symposium on Theory of Computing (STOC)*, pages 479–489, 1989.

[3] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, 1990.

[4] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[5] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *J. Comp. Syst. Sci.*, 18:143–154, 1979.

[6] L. J. Cowen. Compact routing with minimum stretch. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 255–260, 1999.

[7] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proceedings of the 28th International Colloqium on Automata, Languages and Programming (ICALP)*, pages 757–772. Volume 2076 of LNCS, 2001.

[8] Pierre Fraigniaud and Cyril Gavoille. A space lower bound for routing in trees. In 19^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2285 of LNCS, pages 65–75, 2002.

[9] C. Gavoille. Routing in distributed networks: overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001.

[10] C. Gavoille and M. Gengler. Space-efficiency for routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61(5):679–687, 2001.

[11] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2-3):111–120, 2003.

[12] T. Hagerup and P.B. Miltersen, R. Pagh: Deterministic Dictionaries. *J. Algorithms* 41(1): 69-85 (2001).

[13] N. Nisan. $RL \subseteq SC$. In *Proceedings of the 24th annual ACM Symposium on Theory of computing (STOC)*, pages 619–623, 1992.

[14] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[15] D. Sivakumar. Algorithmic derandomization via complexity theory. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing (STOC)*, pages 619–626, 2002.

[16] R. E. Tarjan and A. C. Yao. Storing a Sparse Table. *Commun. ACM* 22(11): 606–611 (1979)

[17] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the 33rd annual ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.

[18] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the 13th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001.