

Compact Routing with Minimum Stretch

Lenore J. Cowen *

Department of Mathematical Sciences

Department of Computer Science

Johns Hopkins University

Baltimore, MD 21218

Abstract

We present the first universal compact routing algorithm with maximum stretch bounded by 3 that uses sublinear space at every vertex. The algorithm uses local routing tables of size $O(n^{2/3} \log^{4/3} n)$ and achieves paths that are most 3 times the length of the shortest path distances for all nodes in an arbitrary weighted undirected network. This answers an open question of Gavoille and Gengler who showed that any universal compact routing algorithm with maximum stretch strictly less than 3 must use $\Omega(n)$ local space at some vertex.

1 Introduction

Let $G = (V, E)$ with $|V| = n$ be a labeled undirected network. Assuming that a positive cost, or distance is assigned with each edge, the stretch of path $p(u, v)$ from node u to node v is defined as $\frac{|p(u, v)|}{|d(u, v)|}$, where $|d(u, v)|$ is the length of the shortest $u - v$ path. The approximate all-pairs shortest path problem involves a tradeoff of stretch against time—short paths with stretch bounded by a constant are computed in time less than it would take to compute exact all-pairs shortest paths (see [1, 2, 6, 8, 9, 10]). The *compact routing* problem considers instead a tradeoff of stretch for space, in the setting where each node locally stores its own routing tables. The *stretch* of a compact routing algorithm is defined as the maximum stretch over the routes for all pairs of nodes in the network. Clearly if each node stores the $O(\log n)$ -bit name of the next node along the shortest path to node v , for all $v \in G$, this complete routing table gives all shortest path distances. The resulting routing scheme uses $O(n \log n)$ space at every node, and has optimal stretch one. This paper answers the question: what is the minimum achievable stretch of any compact routing scheme with sublinear space

at each node? It does so by presenting the first such algorithm with stretch 3. The matching lower bound comes from a recent paper of Gavoille and Gengler [15] who proved that there exists an n -node network that requires $\Omega(n)$ space at some node for any routing scheme with stretch strictly less than 3.

Previous work. Early work on the compact routing problem focused on routing schemes for special case networks such as rings, trees [20], complete networks and grids [21, 22]. Fredrickson and Janardan [12, 13] considered compact routing in networks with small separators, such as outerplanar networks, and also derived small stretch algorithms for compact routing in planar networks. Peleg and Upfal [18, 19] were the first to construct universal compact routing schemes, that is, compact routing schemes that worked on all undirected networks.

Some of the earlier work on this problem presented schemes that bounded only *average* space, rather than maximum space at each node. An algorithm of [5] achieves the optimum of stretch 3 and $O(n^{3/2} \log n)$ space total, but some individual nodes use $O(n \log n)$ space in this scheme. Further work (see [4]) argued successfully that it was most important to bound the maximum space, and this has been the focus of recent work. The best bounds on maximum local space were achieved for different ranges of stretch by different papers: A recent algorithm of [11] achieves $O(n^{1/2} \log n)$ maximum local space with maximum stretch bounded by 5; the paper of [7] give tradeoffs of stretch versus space that result in sublinear local space when the maximum stretch is ≥ 16 , and beat the space requirements of [11] for maximum stretch > 64 . No algorithm which beat the $O(n \log n)$ local space of the naive algorithm for stretch < 5 was previously known. This work presents over a $2/3$ reduction in the quality of the approximate solution, which is now tight for stretch.

*Supported in part by ONR grant N00014-96-1-0829. Author's email: cowen@cs.jhu.edu

Our results. Our algorithms use node names of $O(\log n)$ size, local space bounded by $O(n^{2/3} \log^{4/3} n)$ at every node, and provides paths of maximum stretch 3. In addition, our routing scheme is memoryless— a packet need not remember its origin; the route it takes from intermediate node i is simply a function of the routing information stored at i and the address of its final destination.

The principal ingredients of our algorithm include the following:

- The $O(\log n)$ greedy approximation to dominating set, coupled with truncated and full Dijkstra’s algorithms as used in [1, 2, 9, 10] and in the same fashion as to how it is used in [4].
- A new density dependent algorithm for landmark selection.

It is possible to give a careful distributed implementation of our algorithms, along the same lines as in [5]. The improvements we present are at the algorithmic, not at the protocol level, and go through in the same distributed model with no additional complication. The reader is also referred to the excellent survey of [14]. For those unfamiliar with past work in the area, however, it is worth making a remark, about node-names and output edge-names, both of which are implementation issues common to all papers in the area.

For reasons of symmetry-breaking, typically in the distributed environment, nodes are assumed to be assigned unique $O(\log n)$ -bit names, for example from the integers $\{1, \dots, n\}$. Each node v is also assumed to have a unique name from $\{1, \dots, \deg(v)\}$ assigned to each outgoing edge, but these names are assumed to be assigned locally with no global consistency. As an illustrative example, suppose u and v are adjacent, and say u is assigned the unique node name 1 and v is assigned the unique node name 5. However, u may call its link to v port 200 and v may call assign its link to u port 1080, where these numbers have no relation to 1 and 5. In addition, v may have another link which it calls port 200, but this might go to a different vertex y !

The naive scheme which uses $O(n \log n)$ bits at each node can use these arbitrary names without modification: each node simply stores a table which says for each destination v , which port corresponds to an edge along the shortest path to v . With the exception of the naive scheme (and also [4], see below), all compact routing schemes including ours that guarantee some upper bound on maximum stretch begin with the assignment of a new $O(\log n)$ bit label to every node, this time de-

pendent on network topology— whether this is referred to as a “renaming” or as an “address” is dependent on the particular terminology used in the paper. (The paper of [4] uses topology-independent node names, by putting a distributed dictionary on top of the renaming scheme, so that packets learn the address of the node they are looking for online, as they explore the graph, but to achieve this they need a stronger model: they require writable headers so that packets can be modified en route to store the new address once discovered. Their algorithms also require a larger stretch than we tolerate here.)

We also remark that the re-addressing scheme used by our algorithm is extremely simple: a nodes address becomes a triple consisting of: (it’s original node name, the original name of another node chosen from a special set of nodes called “landmarks”, and finally, the name of the first link on the shortest path from the chosen landmark to the node). Thus our algorithm uses $3 \log n$ -bit node address-names, which is three times the lower-bound required to give every node a unique name. The local port names that each node assigns to its outgoing edges stay unmodified.

2 Overview of the Algorithm

Intuitively, the algorithm bears a great deal of resemblance to how directions are typically given in the real world. Each locale considers some nodes to be near, or local, and some to be far away. For local destinations, detailed information giving exact optimal routes is stored, and for non-local far destinations, optimal routing information is only stored to a subset of these destinations called “landmarks”.

Suppose now there is a yard sale at a given location, and the host wishes to announce it in the newspaper. The host simply publishes his address, which is simply an efficient encoding of his original name and the name of his closest landmark. For nearby nodes, the address can be looked up directly in the local routing table to provide the optimal route. For nodes outside the local neighborhood, instead the optimal route to the landmark is retrieved, and the planned route is to the landmark, and then from the landmark to the destination. Routing to the landmark, rather than directly to the destination may take one out of the way, but because the origin node was far from the destination, landmarks can be chosen in such a way that the extra distance incurred by the potential detour through the landmark is not that large compared to the optimal distance.

The idea of using landmarks to handle routing to “far” destinations is not new with this paper, and is indeed used in several of the previous papers. In fact, the “far” distances are somehow the easy ones: when stretch > 1 is allowed, if the optimal distance between a pair of nodes is large, then large too is the amount out of the way that one is allowed to detour, while still maintaining a stretch bounded by a small multiple of the optimal distance. Previous schemes fell short of achieving stretch 3 or sublinear space at every node because of how they dealt with the “near” distances, or local routing. Our algorithm uses a two-tiered method of selecting landmarks, and a density dependent notion of near and far distances in order to achieve its bounds, as described in Section 4.

3 Preliminaries

The algorithm uses the following known theorem for landmark selection (see [4, 16]).

THEOREM 3.1. (EXTENDED DOMINATING SET)
Let $G = (V, E)$ be a weighted undirected graph. Let B_v denote the set of v 's n^α closest neighbors (where ties are broken lexicographically according to original node names). Then there exists a set $D \subset V$ such that $|D| = O(n^{1-\alpha} \log n)$ and $\forall v \in V, D \cap B_v \neq \emptyset$. Furthermore D can be found by a greedy algorithm in $\tilde{O}(m + n^{1+\alpha})$ time. \square

We also use the following notation. In the course of the algorithm, we will use l_v to denote the landmark closest to v . We will denote by $e_u(v)$ the local name given by u to the outgoing port that corresponds to the first edge on the shortest path from u to v .

We are now ready to describe the main algorithm.

4 The routing algorithm.

The routing algorithm consists of three parts: a labeling algorithm, where nodes are assigned addresses, a storage algorithm, which describes the routing tables stored at each node, and the routing algorithm itself, which describes, based on the address of a destination and the local routing table at the current location, to which output port a packet gets sent.

4.1 The labeling algorithm. Assume $G = (V, E)$ is an connected undirected network with n nodes, m edges, and positive edge weights, and assume the nodes $v \in V$ have been given unique initial identifiers from the set $\{1, \dots, n\}$, that we will denote by $ID(v)$, or

sometimes, if it is clear, simply by v . In the case of where a set is defined in terms of a node's k closest neighbors, ties in distance are broken according to node names: if $d(u, v) < d(u, r)$ then v is closer to u than r . If $d(u, v) = d(u, r)$ and $ID(v) < ID(r)$, then define v as closer to u than r . The labeling algorithm computes the n^α closest neighbors of each node v , according to this ordering, call this v 's *ball*. It then constructs the set of landmarks in two parts, one which is an extended dominating set (as in Theorem 3.1) for the collection of balls, and the second consisting of all nodes which lie in $> n^{(1+\alpha)/2}$ balls. Now each node v locates its closest landmark, and sets its name to be its ID, followed by the ID of that landmark, followed by the landmark's local name for its link corresponding to the first edge on the shortest path from u to v . The labeling algorithm is described fully in Figure 1.

4.2 The storage algorithm. Recall $e_u(v)$ denotes the port at u that leads to the first edge on the shortest path from node u to node v . Thus the naive scheme stores $(v, e_u(v))$ for all nodes v at u . Our scheme will store a subset of the same information at u as follows:

- For each landmark l , store $(l, e_u(l))$
- For each v s.t. v is closer to u than any landmark, store $(v, e_u(v))$

Running full Dijkstra from each of the landmarks computes $(l, e_u(l))$, and the remaining entries $(v, e_u(v))$ can be computed using truncated-Dijkstra from u to its n^α closest nodes (keeping track of when one goes through an existing landmark); since $D \subset L$ covers B_u , all nodes which are not reached by truncated-Dijkstra are guaranteed to be further than some landmark from u . The storage algorithm is summarized in Figure 2.

THEOREM 4.1. *In an n -node m -link network, the labeling and local storage tables can be computed in $\tilde{O}(n^{1+2\alpha} + n^{1-\alpha}m + n^{(1+\alpha)/2}m)$ time.*

Proof. Truncated Dijkstra takes $\tilde{O}(n^{2\alpha})$ time (see [10]), since the labeling and storage algorithms each perform truncated Dijkstra for all n vertices, this takes a total of $\tilde{O}(n^{1+2\alpha})$ time. Construction of L is dominated by the construction of D which takes $\tilde{O}(m + n^{1+\alpha})$ time. The rest of the labeling and storage tables are computed from running full Dijkstra from each $l \in L$, at a cost of $\tilde{O}(m|L|)$, where $|L|$ will be shown to be $O(n^{1-\alpha} \log n + n^{\frac{1+\alpha}{2}})$ by Lemma 5.4 below. \square

We remark that the labeling and storage algorithms (which form the pre-processing for our routing algo-

```

For each  $v \in V$ , perform truncated-Dijkstra( $n^\alpha$ )
and store:
   $B_v \leftarrow \{y | y \text{ is one of } v\text{'s } n^\alpha \text{ closest neighbors.}\}$ 
   $R_v \leftarrow \{y | v \in B_y\}$ 
 $C \leftarrow \{v | |R_v| > n^{\frac{1+\alpha}{2}}\}$ 
 $D \leftarrow \{\text{The extended dominating set of Theorem 3.1.}\}$ 
 $L \leftarrow C \cup D$  /*  $L$  is the set of landmarks */
For each  $v \in V$ 
   $l_v \leftarrow \arg \min_{l \in L} d(l, v)$ 
For each  $l \in L$  perform truncated-Dijkstra( $n^\alpha$ )
For each  $v \in V \setminus L$ 
   $v \leftarrow (v, l_v, e_{l_v}(v))$  /* the first link on the shortest path from  $l_v$  to  $v$  */

```

Figure 1: The labeling algorithm

```

For each  $v \in V$ , perform truncated-Dijkstra( $n^\alpha$ )
(Keeping track when node  $u$  is reached, whether  $\exists l \in L$  on the
shortest path from  $v$  to  $u$ .)
  For each  $u$  reached by truncated-Dijkstra for  $v$ 
    If  $\nexists l \in L$  on shortest path from  $v$  to  $u$ 
      Store  $(v, e_u(v))$  at  $u$ 
  For each  $l \in L$ , perform full-Dijkstra with source  $l$ 
    For each  $u \in V$ 
      Store  $(l, e_u(l))$  at  $u$ 

```

Figure 2: The storage algorithm

algorithm) resemble the approximate shortest path algorithms of [10]. In fact, we are constructing a 3-*spanner* (see [3, 10, 17]). However, we are doing more work than the best tradeoffs achieved by those other algorithms, because our goal is not only to produce approximate shortest paths, but to have small local *space*. Thus we employ more landmarks (and run the expensive full Dijkstra more times) than these approximate shortest path algorithms. Nonetheless, our storage and labeling algorithms, together with the routing algorithm, present a class of subcubic algorithms for approximate all-pairs shortest paths, for all $0 < \alpha < 1$.

4.3 Routing using the local tables We are now ready for the routing procedure. Let the network be labeled and routing tables stored according to the algorithms of the previous section. We give a memory-less algorithm (i.e. packets need only know their destination, they do not have to remember previous state information, such as where they came from) for routing from any network node to node v , with a stretch bounded by 3. We note that the only previous algorithm that has the memory-less property is the naive scheme. All other

known compact routing schemes require writable headers to store state information at intermediate nodes.

4.4 The routing algorithm. The following procedure is used to route at node u , a packet with destination $(v, l_v, e_{l_v}(v))$.

- If $u = l_v$ route along $e_{l_v}(v)$.
- If not, but $(v, e_u(v))$ is in u 's local routing table, route along $e_u(v)$.
- Else route along $(l_v, e_u(l_v))$.

5 Analysis of the algorithm

We first need the following lemmas.

LEMMA 5.1. *If $d(u, v) < d(l_v, v)$ then u is not a landmark and there does not exist a landmark on the shortest path from u to v .*

Proof. If $x \in L$ is u or an intermediate vertex on the shortest path from u to v , then $d(x, v) \leq d(u, v) <$

$d(l_v, v)$, so x is a closer landmark to v than l_v , violating the definition of l_v . \square

LEMMA 5.2. *For each v , l_v is among v 's n^α closest neighbors.*

Proof. For each v there exists some landmark $l \in L$ among v 's n^α closest neighbors because $D \subset L$ and $D \cap B_v \neq \emptyset$. By definition $d(l_v, v) \leq d(l, v)$ so l_v is also among v 's n^α closest neighbors. \square

LEMMA 5.3. *For each $x \neq l_v$ on the shortest path from l_v to v , $(x, e_x(v))$ is stored at x .*

Proof. By Lemma 5.2, l_v is among v 's n^α closest neighbors. Since $d(x, v) < d(l_v, v)$ for each $x \neq l_v$ on the path, it must be both that $x \notin L$ (since l_v is v 's closest landmark) and x is also among v 's n^α closest neighbors. But this is exactly the condition when the storage algorithm stores $(x, e_x(v))$ at x . \square

LEMMA 5.4. $|L| = O(n^{1-\alpha} \log n + n^{\frac{1+\alpha}{2}})$

Proof. $|L| = |C \cup D| \leq |C| + |D|$. Now $|D| = O(n^{1-\alpha} \log n)$ by Theorem 3.1. Recall that $C = \{y \mid |R_y| > n^{\frac{1+\alpha}{2}}\}$. Since $\sum_y |R_y| = \sum_v |B_v| = \sum_v n^\alpha = n^{1+\alpha}$, the number of $y \in C$ can be at most $n^{\frac{1+\alpha}{2}}$. \square

We are ready to prove the main theorems.

THEOREM 5.1. *Let $d(u, v)$ denote the length of the shortest path from u to v . Then the routing algorithm returns a path of length at most $3d(u, v)$.*

Proof. Let $\text{route}(u, v)$ denote the route taken by the algorithm from u to v . Suppose first that $d(u, v) < d(l_v, v)$. Then for each x on the shortest path from u to v , $d(x, v) < d(u, v) < d(l_v, v)$. Then by Lemma 5.2, $l_v \in B_v$, so $u \in B_v$ and $x \in B_v$ for all x on the shortest path from u to v . Thus, by definition, $v \in R_u$ and $x \in R_u$ for all x on the shortest path from u to v . Furthermore, by Lemma 5.1 none of the vertices on the shortest path from u to v (including u) can be $\in L$. Thus $(v, e_u(v))$ was placed in u 's local routing table, as was $(v, e_x(v))$ for each $x \neq u$ on the shortest path from u to v . Thus $\text{route}(u, v)$ takes the optimum shortest path from u to v and the length of $\text{route}(u, v) = d(u, v)$.

Otherwise, $d(u, v) \geq d(l_v, v)$. If $(v, e_u(v))$ was placed in u 's local routing table, then as before, so was $(v, e_x(v))$ for each intermediate node on the shortest path from u to v , and we route optimally as before. Otherwise, there are two cases. The first is that there is no intermediate node x such that $(v, e_x(v))$ was placed in x 's local routing table on the shortest path from u to l_v (or u is itself l_v) and the second case is that

there is such a node x . In the first case, the algorithm routes along the shortest path from u to l_v , at node l_v it takes the first edge along the shortest path from l_v to v , and then continues from l_v to v along the shortest path by Lemma 5.3. Thus the length of $\text{route}(u, v)$ is precisely $d(u, l_v) + d(l_v, v)$. By the triangle inequality, $d(u, l_v) \leq d(u, v) + d(v, l_v)$, and $d(v, l_v) \leq d(u, v)$ by assumption. Thus $\text{route}(u, v) \leq d(u, l_v) + d(l_v, v) \leq d(u, v) + 2d(v, l_v) \leq 3d(u, v)$.

For case 2, let y be the first node on the shortest path from u to l_v with $(v, e_y(v))$ was placed in y 's local routing table. Then, as before, $(v, e_x(v))$ is placed in x 's local routing table for each intermediate node x on the shortest path from y to v . Thus the algorithm routes along the shortest path from u to y , followed by the shortest path from y to v , at a cost of $d(u, y) + d(y, v)$. Compare this to the length of the route $d(u, l_v) + d(l_v, v) = d(u, y) + d(y, l_v) + d(l_v, v)$ followed in case 1, and the triangle inequality implies routing directly from y to v is always a shortcut. Thus in this case too, the total $\text{route}(u, v)$ is $\leq 3d(u, v)$ as before. \square

THEOREM 5.2. *The local storage space used at each node is $O((n^{1-\alpha} \log n + n^{\frac{1+\alpha}{2}}) \log n)$.*

Proof. Each node stores an entry in its routing table for the first link along the shortest path from it to each landmark, the number of such entries is equal to the number of landmarks in the network, which is $O(n^{1-\alpha} \log n + n^{\frac{1+\alpha}{2}})$, by Lemma 5.4. If v is a landmark, it has no additional storage. Otherwise, v must have $|R_v| \leq n^{\frac{1+\alpha}{2}}$, since otherwise v is made a landmark by definition of C . Thus if v is not a landmark, its table has at most R_v additional entries which is at most $O(n^{\frac{1+\alpha}{2}})$ entries. Since each entry consists of an $O(\log n)$ node and port address, the result follows. \square

COROLLARY 5.1. *There is a compact routing algorithm of maximum stretch 3, with $O(n^{2/3} \log^{4/3} n)$ storage at each node.*

Proof. Follows from the previous two theorems, setting $\alpha = 1/3 + \frac{2 \log \log n}{3 \log n}$. \square

References

- [1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). Unpublished manuscript.
- [2] D. Aingworth, C. Chekuri, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 547–553, Atlanta, Georgia, 28–30 Jan. 1996.

- [3] B. Awerbuch. Complexity of network synchronization. *J. of the ACM*, 32(4):804–823, Oct. 1985.
- [4] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive network routing. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 479–489, May 1989.
- [5] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Improved routing strategies with succinct tables. *J. of Algorithms*, 11:307–341, 1990.
- [6] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM J. on Comput.*, 28(1):263–277, 1999.
- [7] B. Awerbuch and D. Peleg. Routing with polynomial communication - space trade-off. *SIAM J. Disc. Math*, 5(2):151–162, 1992.
- [8] E. Cohen. Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM J. on Comput.*, 28(1):210–236, 1999.
- [9] E. Cohen and U. Zwick. All-pairs small-stretch paths. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 93–102, New Orleans, Louisiana, 5–7 Jan. 1997.
- [10] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. In *37th Annual Symposium on Foundations of Computer Science*, pages 452–461, Burlington, Vermont, 14–16 Oct. 1996. IEEE.
- [11] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. In *17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 11–20, 1998.
- [12] G. N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, Aug. 1988.
- [13] G. N. Frederickson and R. Janardan. Efficient message routing in planar networks. *SIAM J. on Comput.*, 18:843–857, Aug. 1989.
- [14] C. Gavoille. A survey on interval routing scheme. Technical Memo RR-1182-97, Laboratoire Bordelais de Recherche en Informatique, Oct. 1997.
- [15] C. Gavoille and M. Gengler. Space-efficiency of routing schemes of stretch factor three. In *4th International Colloquium on Structural Informtion and Communication Complexity (SIROCCO)*, pages 162–175, July 1997.
- [16] L. Lovasz. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.
- [17] D. Peleg and A. A. Schäffer. Graph spanners. *J. of Graph Theory*, 13:99–116, 1989.
- [18] D. Peleg and E. Upfal. A tradeoff between size and efficiency for routing tables. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 43–52. ACM, May 1988.
- [19] D. Peleg and E. Upfal. A tradeoff between size and efficiency for routing tables. *J. of the ACM*, 36:510–530, 1989.
- [20] N. Santoro and R. Khatib. Implicit routing in networks. *The Computer Science Journal*, 28:5–8, 1985.
- [21] J. van Leeuwen and R. Tan. Routing with compact routing tables. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 259–273. Springer-Verlag, New York, New York, 1986.
- [22] J. van Leeuwen and R. Tan. Interval routing. *The Computer Journal*, 30:259–273, 1987.