Optimal Routing in Chord

Prasanna Ganesan* Gurmeet Singh Manku†

Abstract

We propose optimal routing algorithms for Chord [1], a popular topology for routing in peer-to-peer networks. Chord is an undirected graph on 2^b nodes arranged in a circle, with edges connecting pairs of nodes that are 2^k positions apart for any $k \geq 0$. The standard Chord routing algorithm uses edges in only one direction. Our algorithms exploit the bidirectionality of edges for optimality. At the heart of the new protocols lie algorithms for writing a positive integer d as the difference of two non-negative integers d' and d'' such that the total number of 1-bits in the binary representation of d' and d'' is minimized. Given that Chord is a variant of the hypercube, the optimal routes possess a surprising combinatorial structure.

1 Introduction

Consider an undirected graph on 2^b nodes arranged in a circle. Nodes are labeled with b-bit identifiers from 0 through $2^b - 1$ going clockwise. An edge (x, y) exists iff x and y are 2^k positions apart on the circle for some $k \geq 0$, i.e., $|x - y|$ equals either 2^k or $2^b - 2^k$ for some $0 \leq k < b$. The resulting graph forms the basis of Chord [1], a topology used for routing in peer-to-peer networks. Nodes in the topology represent computers and edges represent overlay-network connections. Since overlay connections span wide-area networks, a message sent along a connection might incur a potentially large delay. Therefore, it is important to minimize the path length of routes on this overlay network.

In the standard Chord routing algorithm, messages are forwarded along only those edges that diminish the clockwise distance by some power of two. Routing is clockwise and greedy, never overshooting the destination. Thus if a message is destined for a node that is clockwise distance d away, routing is equivalent to performing left-to-right bit-fixing to convert the 1s in the binary representation of d to zero. For example, if d is 14 (1110 in binary), greedy routing uses steps of 8, 4 and 2 in that order, thus converting the leftmost 1 in the remaining distance to a 0 at each step. The longest path has length b and the average path length is $b/2$. Clockwise greedy routing is non-optimal because it fails to exploit the bidirectionality of edges. Consequently, the average path length for this routing protocol is no better than that for a hypercube on as many nodes, even though Chord has roughly twice as many edges.

We develop new routing protocols for Chord that exploit the bidirectionality of edges. It turns out that optimal routes in Chord have a strong connection with the *Binary Subtraction Problem*: Given a positive integer d, find a pair of non-negative integers $\langle d', d'' \rangle$ such that the number of 1-bits in d' and d'' is minimal, subject to the constraint $d = d' - d''$. For example, the shortest route to cover a clockwise distance 14 (1110 in binary) is to use a clockwise step of 16 in combination with an anti-clockwise step of length 2, which can be seen as an optimal way of expressing 14 as the difference of two numbers.

One may wonder why the Chord topology and, consequently, these routing protocols, are relevant in a practical peer-to-peer system where the number of participant computers may not be a power of two. It turns out that it is possible to create small, roughly equi-sized *groups* of computers such that the number of groups is a power of two. The Chord topology is then constructed over these groups and used for routing from one group to another.

Road map: In §2, we study a simple but non-optimal variant of the standard Chord routing algorithm: each node chooses between the shorter of clockwise-greedy and anti-clockwise-greedy routes, on a per-destination basis. We show that the average path length drops to $b/2 - \sqrt{b/(2\pi)} + \Theta(1)$.

In §3, we expose the interplay between optimal routing in Chord and binary subtraction. In §4, we solve the *Binary Subtraction Problem*. In general, there is no unique solution. We present a non-deterministic procedure that generates all the optimal solutions. In §5, we provide optimal routing algorithms for Chord. We show that Chord's diameter is $\lfloor b/2 \rfloor$. However, the average all-pairs shortest-path length is only $b/3 + \Theta(1)$. Interestingly, two simple algorithms that discover optimal routes can be encoded compactly by finite-state

*Department of Computer Science, Stanford University, USA, prasannag@cs.stanford.edu. Supported in part by a Stanford Graduate Fellowship.

†Department of Computer Science, Stanford University, USA, manku@cs.stanford.edu. Supported in part by NSF Grant EIA-0137761 and grants from SNRC and Veritas.

automata. The average shortest-path lengths are then computed by treating the automata as Markov Chains.

In §6, we study link congestion for all the routing algorithms we describe. In §7, we extend our results to higher-base versions of Chord. In §8, we list open problems pertaining to the Hyperskewbe, a rather mysterious topology obtained by subtracting the hypercube edges from Chord but retaining the diameter edges.

2 Choice of Two

A simple variant of the standard Chord routing algorithm is as follows: When node x wishes to send a message to y, it chooses the shorter of

(a) the clockwise greedy route to y, and

(b) the anti-clockwise greedy route to y's clockwise successor (namely $(y+1) \bmod 2^b$), followed by an anti-clockwise step to y.

When both (a) and (b) have equal path lengths, we arbitrarily choose (b). The clockwise-greedy route in (a) considers only those edges that diminish the clockwise distance by some power of two. Similarly, the anti-clockwise-greedy route in (b) considers only those edges that diminish the anti-clockwise distance by some power of two. If we replace (b) by the more natural "anti-clockwise greedy route to y", Theorem 2.1 (see below) remains largely unchanged (the longest path is $\lfloor b/2 \rfloor$ instead of $\lceil b/2 \rceil$). However, congestion analysis in §6 becomes fairly involved.

We now compute the average path length for Choice of Two. Let $d = (y - x + 2^b) \bmod 2^b$, the clockwise distance from x to y. Then the length of the clockwise-greedy route is $H(d)$ where $H(x)$ denotes the Hamming norm of x, i.e., the number of 1-bits in x. The length of the anti-clockwise greedy route, as described above, is $b - H(d) + 1$. The shorter of the two has length $\min\{H(d), b - H(d) + 1\}$. Let

$$S = \sum_{d:H(d)\le b} \min\{H(d), b - H(d)\}$$
$$T = \sum_{i=0}^{b} i\binom{b}{i} = b2^{b-1}$$

The average shortest-path length for clockwise greedy routes is $T/2^b = b/2$. The average for Choice of Two is $S/2^b + \Theta(1)$. The quantity $S/2^b$ can be viewed in terms of a bins-and-balls problem: if each of b balls is thrown at random into one of two identical bins, then $S/2^b$ is the expected size of the smaller bin.

Lemma 2.1. *For $0 \le m < b$*

$$\sum_{j=0}^{m} (b - 2j)\binom{b}{j} = (m+1)\binom{b}{m+1}$$

Proof. By induction.

Theorem 2.1. *The average path length for the* Choice of two *algorithm is $b/2 - \sqrt{b/2\pi} + \Theta(1)$ while the longest path has length $\lceil b/2 \rceil$.*

Proof. The average path length is at most $S/2^b + 1$. If b is odd, $S = S_{odd}$, otherwise $S = S_{even}$, where

$$S_{odd} = \sum_{i=0}^{(b-1)/2} i\binom{b}{i} + \sum_{i=(b+1)/2}^{b} (b-i)\binom{b}{i}$$
$$S_{even} = \sum_{i=0}^{b/2} i\binom{b}{i} + \sum_{i=b/2+1}^{b} (b-i)\binom{b}{i}$$

Then

$$T - S_{odd} = \sum_{i=(b+1)/2}^{b} (2i - b)\binom{b}{i}$$
$$T - S_{even} = \sum_{i=(b+2)/2}^{b} (2i - b)\binom{b}{i}$$

Substituting $j = b - i$, we get

$$T - S_{odd} = \sum_{j=0}^{(b-1)/2} (b - 2j)\binom{b}{j}$$
$$T - S_{even} = \sum_{j=0}^{(b-2)/2} (b - 2j)\binom{b}{j}$$

Using Lemma 2.1, we get

$$T - S_{odd} = \frac{(b+1)}{2}\binom{b}{(b+1)/2}$$
$$T - S_{even} = (b/2)\binom{b}{b/2}$$

We use Stirling's approximation, which is

$$\sqrt{2\pi x}(x/e)^x e^{1/(12x+1)} < x! < \sqrt{2\pi x}(x/e)^x e^{1/12x}$$

to deduce

$$T - S_{odd} = \left[\sqrt{(b+1)/2\pi} + \Theta(1)\right] 2^b$$
$$T - S_{even} = \left[\sqrt{b/2\pi} + \Theta(1)\right] 2^b$$

Thus, the average path length is $b/2 - \sqrt{b/2\pi} + \Theta(1)$. The longest path length is $\lceil b/2 \rceil$ corresponding to those values of d with $\lceil b/2 \rceil$ ones and $\lfloor b/2 \rfloor$ zeros. □

3 Optimal Routing and Binary Subtraction

Consider a route from node x to node y in Chord. Let us label each edge in the route by either a positive or a negative power of two as follows: Let c denote the clockwise distance traversed by following some edge. If $c = 2^{b-1}$, we label the edge arbitrarily as -2^{b-1} or $+2^{b-1}$. If $c = 2^k$ for some $0 \le k < b-1$, then the label is $+2^k$. Otherwise, $c = 2^b - 2^k$ for some $0 \le k < b-1$ and the label is -2^k. In other words, we mark clockwise and anti-clockwise usage of edges of length 2^k with positive and negative signs respectively.

Chord is symmetric with respect to every node. Therefore, a route corresponding to one of the shortest paths between x and y enjoys two properties: (a) no two labels along the path sum to zero, and (b) no label appears twice. Let d' be the sum of the positive labels, and d'' the sum of the absolute values of the negative labels. Recall that $H(x)$ is the Hamming norm of x, i.e., the number of ones in the binary representation of x.

Since each 1-bit in the binary representation of d' and d'' corresponds to some edge along the route, the length of the route is $H(d') + H(d'')$. Moreover, either $d = d' - d''$ or $2^b - d = d'' - d'$. To explain, when d' is greater than d'', their difference is the clockwise distance covered, and when d'' is greater than d', their difference is the anti-clockwise distance covered. We are now ready for a formal definition of the Chord routing problem.

Chord Routing Problem: *Given $b \geq 1$ and $0 < d < 2^b$, identify $\langle d', d'' \rangle$ such that $H(d') + H(d'')$ is minimal, subject to two constraints:*

(i) either $d = d' - d''$ or $2^b - d = d'' - d'$.

(ii) both $d', d'' \in [0, 2^b)$.

In general, for fixed b and d, there are several optimal solutions for this problem. For example, if $b = 4$ and $d = 0110$, then $\langle 0110, 0000 \rangle$, $\langle 1000, 0010 \rangle$ and $\langle 0000, 1010 \rangle$ are three different optimal solutions. Moreover, a specific $\langle d', d'' \rangle$ actually encodes a *set* of optimal routes in Chord because any permutation of edges corresponding to 1s in $\langle d', d'' \rangle$ covers clockwise distance d.

It turns out that the Chord Routing Problem can be reduced to the following Binary Subtraction Problem:

Binary Subtraction Problem : *Given positive integer d, find a pair of non-negative integers $\langle d', d'' \rangle$ such that $H(d') + H(d'')$ is minimized, subject to the constraint $d = d' - d''$.*

Both Binary Subtraction and Chord Routing produce a tuple $\langle d', d'' \rangle$ as output. However, the two problems differ in two respects, both involving the number of bits b. First, Binary Subtraction has only one constraint $d = d' - d''$ whereas Chord Routing has a choice: either $d = d' - d''$ or $2^b - d = d'' - d'$. Second, in Chord Routing, both d' and d'' are restricted to lie in the range $[0, 2^b)$. There is no such restriction in Binary Subtraction.

Procedure Opt_Route(b, d) below shows how the Chord Routing Problem can be solved by using procedure Opt_Subtract(d), which solves the Binary Subtraction Problem.

```
PROCEDURE OPT_ROUTE(b, d)
    IF (d < 2^(b-1))
        ⟨d', d''⟩ ← OPT_SUBTRACT(d)
        OUTPUT ⟨d', d''⟩
    ELSE
        ⟨d', d''⟩ ← OPT_SUBTRACT(2^b − d)
        OUTPUT ⟨d'', d'⟩
```

The next two Lemmas establish the correctness of procedure Opt_Route. We will discuss procedure Opt_Subtract in detail in §4.

Lemma 3.1. *If $\langle d', d'' \rangle$ = Opt_Subtract(d) and $1 \leq d \leq 2^{b-1}$, then $\langle d', d'' \rangle$ is a valid solution for the Chord Routing Problem with parameters b and d.*

Proof. The proof is in two parts: First, we show that when $1 \leq d \leq 2^{b-1}$, then constraint *(i)* in the definition of the Chord Routing Problem can be simplified by dropping the latter condition. Second, we show that constraint *(ii)* can be dropped altogether. The resulting problem is simply the Binary Subtraction Problem with input d.

(a) Let $\mathcal{S}(b, x)$ denote the set of all tuples $\langle x', x'' \rangle$ such that $x = x' - x''$ and both $x', x'' \in [0, 2^b)$. We will now show that if $d \leq 2^{b-1}$, then for any $\langle d', d'' \rangle \in \mathcal{S}(b, 2^b - d)$, $\langle 2^{b-1} + d'', d' - 2^{b-1} \rangle \in \mathcal{S}(b, d)$. We will also that $H(d') + H(d'') = H(2^{b-1} + d'') + H(d' - 2^{b-1})$ Together, the two claims will prove that for $1 \leq d \leq 2^{b-1}$, we can simplify constraint *(i)* in the Chord Routing Problem by dropping the second condition.

Consider $\langle d', d'' \rangle \in \mathcal{S}(b, 2^b - d)$. By definition, $2^b - d = d' - d''$. Re-arranging the terms, we get $d = (2^{b-1} + d'') - (d' - 2^{b-1})$. By assumption, $d \leq 2^{b-1}$. Therefore, $2^b - d \geq 2^{b-1}$, in which case, $d' \geq 2^{b-1}$ and $d'' < 2^{b-1}$. In other words, the "diagonal" edge is necessarily a part of d' and definitely not in d''. Now, consider the tuple $\langle 2^{b-1} + d'', d' - 2^{b-1} \rangle$. Loosely speaking, this tuple can be obtained from $\langle d', d'' \rangle$ by "flipping" the direction of the diagonal. Since $d' \geq 2^{b-1}$ and $d'' < 2^{b-1}$, it follows that both members of the tuple $\langle 2^{b-1} + d'', d' - 2^{b-1} \rangle$ lie in the interval $[0, 2^b)$. Thus $\langle 2^{b-1} + d'', d' - 2^{b-1} \rangle \in \mathcal{S}(b, d)$. It is easy to see that $H(d') + H(d'') = H(d' - 2^{b-1}) + H(2^{b-1} + d'')$.

(b) Since $d \leq 2^{b-1}$, if $\langle d', d'' \rangle$ is a solution of Opt_Subtract(d), then both $d', d'' \in [0, 2^b)$. Therefore, constraint *(ii)* in the Chord Routing Problem can be dropped. □

Lemma 3.2. *If $\langle d', d'' \rangle$ is a solution for the Chord Routing Problem with parameters b and d, then $\langle d'', d' \rangle$ is a solution for the Chord Routing Problem with parameters b and $2^b - d$.*

Proof. Follows from the problem definition. □

Next, we present the solution to the Binary Subtraction Problem in §4. In §5, we will derive two simple, intuitive algorithms to directly discover optimal routes in Chord. In fact, these algorithms are simple enough to be described as finite-state automata.

4 Solving the Binary Subtraction Problem

Figure 1 shows pseudo-code for OPT_SUBTRACT(d), which solves the Binary Subtraction Problem. The output tuple $\langle d', d'' \rangle$ is produced bit-by-bit, right to left. In fact, OPT_SUBTRACT is non-deterministic and can produce *all* optimal pairs for a given value of input d.

```
PROCEDURE OPT_SUBTRACT (d)
WHILE (d > 0) {

    WHILE (d mod 2 = 0) {
        d ← d/2
        OUTPUT ⟨0, 0⟩
    }

    t := { ⟨1,0⟩          if SUFFIX(d, 0(01)*01)
         { ⟨0,1⟩          if SUFFIX(d, 1(10)*11)
         { ⟨1,0⟩ or ⟨0,1⟩ otherwise

    d := { (d − 1)/2   if t = ⟨1, 0⟩
         { (d + 1)/2   if t = ⟨0, 1⟩

    OUTPUT t
}
```

Figure 1: OPT_SUBTRACT(d) is a non-deterministic procedure that solves the Binary Subtraction Problem. It outputs $\langle d', d'' \rangle$ where $d = d' - d''$ and $H(d') + H(d'')$ is minimal.

Procedure OPT_SUBTRACT effectively *scans* the binary representation of d, right to left, producing the corresponding bits of $\langle d', d'' \rangle$ along the way. As long as the current bit is 0, the output is $\langle 0, 0 \rangle$ since both bits at that position ought to be 0 in $\langle d', d'' \rangle$ for optimality. The current bit becomes 1 when $d \bmod 2 = 1$. Then the output must be either $\langle 1, 0 \rangle$ or $\langle 0, 1 \rangle$. Depending upon whether we choose $\langle 1, 0 \rangle$ or $\langle 0, 1 \rangle$, the algorithm continues scanning the remainder, i.e., $(d-1)/2$ or $(d+1)/2$ respectively. The crux of the matter lies in deciding which of the two possible outputs to produce when $d \bmod 2 = 1$. OPT_SUBTRACT provides a precise characterization of how the decision should be made. The decision involves regular expressions [2].

SUFFIX($x, \mathcal{R}$) denotes a predicate that takes a positive integer x and regular expression $\mathcal{R}$ as input. Let $\mathbf{x}$ denote the binary representation of x in the form of a string with a leading 1. Then SUFFIX($x, \mathcal{R}$) evaluates to true iff string $00\mathbf{x}$ satisfies regular expression $(0+1)^*\mathcal{R}$. In other words, SUFFIX checks whether some suffix of $00\mathbf{x}$ satisfies $\mathcal{R}$. Prepending a pair of zeros to $\mathbf{x}$ might seem unnatural. However, it simplifies proofs by reducing the number of cases that we have to consider.

We now proceed to prove that this algorithm produces optimal solutions for the Binary Subtraction Problem. The reader interested in the algorithms for optimal routing in Chord, rather than in proofs of their optimality, may wish to proceed directly to §5.

LEMMA 4.1. *If $x \bmod 2 = 1$ then exactly one of the following four predicates is true:*

(a) SUFFIX($x, 0(01)^*01$) *(c)* SUFFIX($x, 00(10)^*11$)
(b) SUFFIX($x, 11(01)^*01$) *(d)* SUFFIX($x, 1(10)^*11$)

Proof. SUFFIX($x, \mathcal{R}$) is true iff string $00x$ satisfies $\mathcal{R}$. If $x \bmod 2 = 1$, then exactly one of the following two mutually exclusive conditions holds:

$00x$ *ends in* 01: Either (a) or (b) is true, depending upon whether a pair of zeros or a pair of ones is encountered when scanning $00x$ from right to left.

$00x$ *ends in* 11: Either (c) or (d) is true, depending upon whether a pair of zeros or a pair of ones is encountered first when scanning $00x$ from right to left, and ignoring the right-most bit. □

The non-determinism of OPT_SUBTRACT stems from the fact that we assign $t := \langle 0, 1 \rangle$ or $t := \langle 1, 0 \rangle$ arbitrarily if the suffix of d is neither 0 nor $0(01)^*01$ nor $1(10)^*11$. In other words, whenever d satisfies conditions (b) or (c) in the above lemma, the algorithm has a choice in producing its output. For example, any of the following five outputs can be produced for input d = 1110011010:

d'	10000011010	10000100010
$-d''$	- 00010000000	- 00010001000
d	1110011010	1110011010

10000100000	10000000010	10000000000
- 00010000110	- 00001101000	- 00001100110
1110011010	1110011010	1110011010

4.1 Proof of Optimality We begin by defining functions Φ and G over the set of positive integers.

Φ: For $d \geq 1$, $\Phi(d) = \min[H(d') + H(d'')]$ over all possible $d', d'' \geq 0$ such that $d = d' - d''$.

G: We first define G for strings that satisfy the regular expression $1(1 + 0)^*$. Later, we will extend the definition to include positive integers. For string $\mathbf{x}$ that satisfies regular expression $(1^+0)^+1$, let g denote the number of "groups" of 1s in x; if all groups are singleton ones, $G(\mathbf{x}) = g$, otherwise, $G(\mathbf{x}) = g + 1$. For example, $G(110111) = 2 + 1 = 3$, $G(101) = 2$ and $G(10111) = 2 + 1 = 3$. For any $\mathbf{x}$ that satisfies $(0 + 1)^*1(0 + 1)^*$, we claim that it is possible to write $\mathbf{x} = (0^*)\sigma_1(000^*)\sigma_2(000^*)\ldots\sigma_\ell(0^*)$ uniquely, where each sub-string $\sigma_1, \sigma_2, \ldots, \sigma_\ell$ satisfies the regular expression $(1^+0)^+1$. We are essentially breaking up the

string $\mathbf{x}$ whenever we encounter consecutive zeros, and we are ignoring leading and trailing strings of zeros. Now, $G(\mathbf{x}) = \sum_{i=1}^{\ell} G(\sigma_i)$. For positive integer x, $G(x) = G(\mathbf{x})$ where $\mathbf{x}$ denotes the string representing x in binary.

THEOREM 4.1. *For $d \geq 1$, $\Phi(d) = G(d)$. If $\langle d', d'' \rangle$ is produced by* OPT_SUBTRACT(d) *as output, then $d = d' - d''$ and $H(d') + H(d'') = G(d)$.*

Proof. Proof by induction on integer d. *Base case:* The claim is true for $d = 1$. *Induction step:* Consider $d > 1$. We assume that the claim is true for all integers less than d. There are two cases:

- $d \bmod 2 = 0$

  Clearly, $\Phi(d) = \Phi(d/2)$ and $G(d/2) = G(d)$. By induction hypothesis, $\Phi(d/2) = G(d/2)$. Thus $\Phi(d) = G(d)$. OPT_SUBTRACT indeed outputs $\langle 0, 0 \rangle$ when $d \bmod 2 = 0$.

- $d \bmod 2 = 1$

  If $d = d' - d''$, then exactly one of d' and d'' has 1 in the last bit. Therefore, for $d > 1$,

  $$\Phi(d) = 1 + \min[\Phi((d-1)/2), \Phi((d+1)/2)]$$

  By induction hypothesis,

  $$\Phi((d-1)/2) = G((d-1)/2)$$
  $$\Phi((d+1)/2) = G((d+1)/2)$$

  Therefore,

  $$\Phi(d) = 1 + \min[G((d-1)/2), G((d+1)/2)] \quad (4.1)$$

  The following identities are easily seen to hold:

  a) SUFFIX$(x, 0(01)^*01) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) \end{cases}$

  b) SUFFIX$(x, 11(01)^*01) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) - 1 \end{cases}$

  c) SUFFIX$(x, 1(10)^*11) \Rightarrow \begin{cases} G((x-1)/2) = G(x) \\ G((x+1)/2) = G(x) - 1 \end{cases}$

  d) SUFFIX$(x, 00(10)^*11) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) - 1 \end{cases}$

  From Lemma 4.1, exactly one of the above four SUFFIX predicates must be true for d. Therefore,

  $$\min[G((d+1)/2), G(d-1)/2)] = G(d) - 1 \quad (4.2)$$

  Eq (4.1) and Eq (4.2) together yield $\Phi(d) = G(d)$.

The correctness of OPT_SUBTRACT is clear if we rewrite the assignment to t as follows:

$$t := \begin{cases} \langle 1, 0 \rangle & \text{if } G((d-1)/2) < G((d+1)/2) \\ \langle 0, 1 \rangle & \text{if } G((d-1)/2) > G((d+1)/2) \\ \langle 1, 0 \rangle \text{ or } \langle 0, 1 \rangle & \text{if } G((d-1)/2) = G((d+1)/2) \end{cases}$$

In fact, it can be shown that OPT_SUBTRACT(d) produces *all* possible tuples $\langle d', d'' \rangle$ such that $d = d' - d''$ and $H(d') + H(d'') = G(d)$. □

5 Optimal Routing Algorithms for Chord

We present two deterministic algorithms for solving the Chord Routing Problem that we defined in §3. For fixed values of b and d, exactly one output tuple $\langle d', d'' \rangle$ is produced. Both algorithms run the automaton in Figure 2 for exactly b steps.

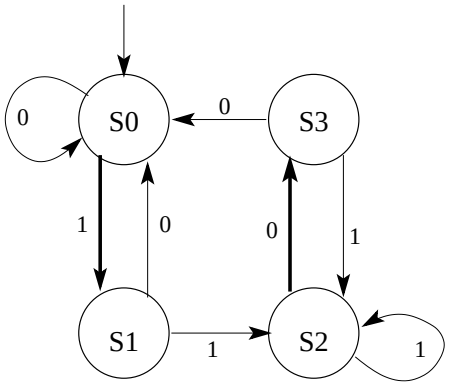

Figure 2: The state machine used by automata-based algorithms that solve the Chord Routing Problem.

5.1 RIGHT-TO-LEFT CHAINING: The binary representation of distance d, possibly padded with leading 0s to make it exactly b bits long, is fed as input right-to-left. The output tuple $\langle d', d'' \rangle$ is also generated right-to-left. The start state is S_0. Each transition produces a pair of bits as output, the first bit for d' and the second for d''. All thin edges produce $\langle 0, 0 \rangle$ as output. Thick edges ($S_0 \xrightarrow{1} S_1$ and $S_2 \xrightarrow{0} S_3$) require a *lookahead*: If the next input bit is 0, the output is $\langle 1, 0 \rangle$. If the next input bit is 1, the output is $\langle 0, 1 \rangle$. If there is no next input bit, i.e., the input string just terminated, the output is arbitrarily chosen as $\langle 0, 1 \rangle$ or $\langle 1, 0 \rangle$. Observe that the traversal of a thick edge corresponds to exactly one 1-bit in either d' or d''.

Further intuition into RIGHT-TO-LEFT CHAINING is gained by understanding the algorithm in terms of two ideas: *chain fixing* and *chain coupling*.

Chain fixing is simple: a distance corresponding to a chain of at least two 1s can be covered using a

combination of only two steps: a short backward step and a long forward step. For example, a distance of 7 (000111 in binary) in a 64-node graph can be covered with a forward step of 8 combined with a backward step of 1. Chain fixing applies only to chains of length at least two. A chain consisting of a solitary 1 is handled by taking a single forward step.

Chain coupling comes into play when two chains are separated by a solitary zero. We explain the idea with an example. Let us say we had to cover clockwise distance 238(011101110). We know that the last five bits can be fixed by a backward step of 2 and a forward step of 16. However, we observe that taking the backward step of 2 leads to a distance 240(011110000), in which the bit corresponding to the intended forward step of 16 becomes part of a longer chain of 1s. Thus, instead of a forward step of 16, we can instead use a *backward* step of 16 followed by a forward step of 256 to fix this entire chain of 1s.

The behavior of the finite-state automaton in Figure 2 can be understood in terms of chain-fixing and chain-coupling. The automaton starts in State S_0, moving between S_0 and S_1 to fix singleton ones. Transition $S_1 \xrightarrow{1} S_2$ marks the onset of chain-fixing because two consecutive ones were just encountered in the input. Transition $S_2 \xrightarrow{0} S_3$, followed by $S_3 \xrightarrow{1} S_2$ marks the onset of chain-coupling. Transition $S_3 \xrightarrow{0} S_1$ signals the end of chain-coupling since two consecutive zeroes were just encountered, taking us to the start state S_0. If the input string happens to terminate in state S_1 or S_3, the output can be chosen arbitrarily as $\langle 1, 0\rangle$ or $\langle 0, 1\rangle$. The choice is immaterial because 1s in the most significant bit-positions of d' and d'' correspond to clockwise distances 2^{b-1} and $2^b - 2^{b-1}$ respectively, which are equivalent.

Observe that the automaton solves OPT_SUBTRACT(d) when $d < 2^{b-1}$. The deterministic algorithm is equivalent to replacing the two lines in Figure 1 that assign values to t and d, as follows:

$$t := \begin{cases} \langle 1, 0\rangle & \text{if SUFFIX}(d, 01) \\ \langle 0, 1\rangle & \text{if SUFFIX}(d, 11) \end{cases}$$

$$d := \begin{cases} (d-1)/2 & \text{if } t = \langle 1, 0\rangle \\ (d+1)/2 & \text{if } t = \langle 0, 1\rangle \end{cases}$$

5.2 LEFT-TO-RIGHT BIDIRECTIONAL GREEDY[1]: The idea is simple: Node x chooses the edge that takes the message the closest, in terms of absolute distance on the circle, to y. This algorithm can also be encoded using the automaton in Figure 2. The input string $d[1..b]$ is treated as a bit-array of length b and is processed from `left` to `right`. The start state is S_0. The output is stored in bit-arrays $d'[1..b]$ and $d''[1..b]$, both of which are initialized to all-zeros. The output arrays get altered only when a thick edge is traversed. Let us assume that the thick edge was traversed due to the i^{th} input bit, where $1 \le i \le b$.

$S_0 \xrightarrow{1} S_1$: `if` $(i = b)$ or $(d[i+1] = 0)$
`then set` $d'[i] \leftarrow 1$
`else set` $d'[i-1] \leftarrow 1$

$S_2 \xrightarrow{0} S_3$: `if` $(i = b)$ or $(d[i+1] = 0)$
`then set` $d''[i-1] \leftarrow 1$
`else set` $d''[i] \leftarrow 1$

5.3 Proof of Optimality

LEMMA 5.1. *Let* $\mathbf{d}$ *denote the binary representation of integer* $d \ge 1$. *The automaton traverses exactly* $G(d)$ *thick edges if string* $0\mathbf{d}$ *is fed right-to-left.*

Proof. Let $0\mathbf{d} = 0\sigma_1(00^*0)\sigma_2(00^*0)\sigma_3 \ldots \sigma_\ell(0^*)$ where each σ_i satisfies the regular expression $(1^+0)^+1$. We claim that the number of thick edges traversed when processing $0\mathbf{d}$ is $G(d) = \sum_{i=1}^{\ell} G(\sigma_i)$. It is easier to prove the claim if we rewrite $0\mathbf{d} = (0\sigma_1 0)0^*(0\sigma_2 0)0^*(0\sigma_3 0)0^* \ldots (0\sigma_\ell)0^*$. The automaton traverses no thick edges while processing the sub-strings corresponding to 0^*. For each $0\sigma_i$, the automaton performs one state transition from S_0 to S_1 for each of the initial singleton 1s, one state transition from S_0 to S_1 for the first of a non-singleton group of 1s, and one state transition from S_2 to S_3 at the end of every group of 1s thereafter. Observe that if the input string does not have a leading zero, i.e., the last input bit fed to the automaton is a 1, the automaton could potentially terminate in state S_2 without traversing a thick edge for the last group of 1s. □

LEMMA 5.2. *For* $1 \le d < 2^b$,

$$\begin{aligned} d \le \lfloor 2^b/3 \rfloor &\Rightarrow G(d) = G(2^b - d) - 1 \\ \lfloor 2^b/3 \rfloor < d \le \lfloor 2^{b+1}/3 \rfloor &\Rightarrow G(d) = G(2^b - d) \\ d > \lfloor 2^{b+1}/3 \rfloor &\Rightarrow G(d) = G(2^b - d) + 1 \end{aligned}$$

Proof. For $d = 2^{b-1}$, the Lemma is true. For $d \ge 1$ and $d \ne 2^{b-1}$, the b-bit strings representing d and $2^b - d$ satisfy three properties: (a) the position of the right-most 1 is the same in both strings, (b) all digits to the left of the right-most 1 are complements of each other in the two strings, and (c) there is at least one digit to the left of the right-most 1.

[1]Thanks to Qixiang Sun for suggesting this algorithm.

Consider two copies of the automaton in Figure 2, one scanning the b-bit string representing d and the other scanning the b-bit string representing $2^b - d$, both right-to-left. Both automata reach state S_1 upon consuming the right-most 1. Thereafter, for every digit (0 or 1) processed as input by the first automaton, its complement is processed by the second. It is easy to see that the two automata are in diagonally opposite states at all times. Therefore, both traverse exactly the same number of thick edges when processing b bits. From Lemma 5.1, we conclude that if we were to run both automata for one more step, with an additional 0 as input, the number of thick edges traversed would be exactly the function G over the respective inputs. The only thick transition upon receiving 0 as input corresponds to $S_2 \xrightarrow{0} S_3$. Therefore, $G(d) = G(2^b - d)$ iff the automaton terminates in S_1 or S_3, and $G(d) = G(2^b - d) - 1$ iff the automaton terminates in S_0. Finally, $G(d) = G(2^b - d) + 1$ iff the automaton terminates in S_2. We are now left with the job of characterizing strings that terminate in the four states.

The automaton is in state S_2 when scanning a string right-to-left iff there exists a prefix of the string that satisfies the regular expression $1(01)^*1$. For $1 \leq d < 2^b$, a prefix of the string representing d satisfies the regular expression $1(01)^*1$ iff $d > \lfloor 2^{b+1}/3 \rfloor$. Therefore, the automaton is in state S_2 iff $d > \lfloor 2^{b+1}/3 \rfloor$. Similarly, it can be shown that the automaton is in state S_0 iff $d \leq \lfloor 2^b/3 \rfloor$. The automaton is in state S_1 or S_3 otherwise.

Thus $G(d) = G(2^b - d) + 1$ iff $d > \lfloor 2^{b+1}/3 \rfloor$, and $G(d) = G(2^b - d) - 1$ iff $d \leq \lfloor 2^b/3 \rfloor$. Otherwise, $G(d) = G(2^b - d)$. □

THEOREM 5.1. *An optimal solution to the Chord Routing Problem has path length*

$$\begin{array}{ll} 0 & \text{if } d = 0 \\ G(d) & \text{if } 1 \leq d \leq \lfloor 2^{b+1}/3 \rfloor \\ G(d) - 1 & \text{otherwise} \end{array}$$

Proof. Follows from Lemma 5.2 and the definition of procedure OPT_ROUTE. □

In fact, Lemma 5.2 suggests an improvement to OPT_ROUTE, as shown in Figure 3. This procedure is a complete characterization of the Chord Routing Problem and produces *all* possible optimal solutions for inputs b and d.

THEOREM 5.2. RIGHT-TO-LEFT CHAINING *solves the Chord Routing Problem optimally. The diameter of Chord is* $\lfloor b/2 \rfloor$.

```
PROCEDURE OPT_ROUTE(b, d)
    IF (d ≤ ⌊2^b/3⌋)
        ⟨d', d''⟩ ← OPT_SUBTRACT(d)
        OUTPUT ⟨d', d''⟩
    ELSE IF (d > ⌊2^{b+1}/3⌋)
        ⟨d', d''⟩ ← OPT_SUBTRACT(2^b − d)
        OUTPUT ⟨d'', d'⟩
    ELSE
        ⟨d', d''⟩ ← { OPT_SUBTRACT(2^b − d)
                    { or
                    { OPT_SUBTRACT(d)
        OUTPUT ⟨d', d''⟩
```

Figure 3: OPT_ROUTE(d) is a non-deterministic procedure that solves the Chord Routing Problem.

Proof. The path length of a route produced by RIGHT-TO-LEFT CHAINING for distance d is exactly equal to the number of thick edges traversed when scanning the binary representation of d right-to-left. As outlined in the proof of Lemma 5.1, RIGHT-TO-LEFT CHAINING traverses thick edges exactly $G(d)$ times iff $G(d) \leq \lfloor 2^{b+1}/3 \rfloor$. Otherwise, it traverses thick edges $G(d) - 1$ times. From Theorem 5.1, we see that this is optimal.

One of the strings that requires the maximum number of steps is $d = (01)^{b/2}$ when b is even and $d = (01)^{(b-1)/2}0$ when b is odd. For both cases, $G(d) = \lfloor b/2 \rfloor$. □

The proof of Theorem 5.2 sheds light on the reason for the automata for both RIGHT-TO-LEFT CHAINING and LEFT-TO-RIGHT BIDIRECTIONAL GREEDY being identical in structure: the automaton is essentially a computation of G over the input string. The automaton remains the same irrespective of whether strings are scanned right-to-left or left-to-right.

5.4 Average Path Length Theorem 5.1 characterizes the path length of optimal solutions for the Chord Routing Problem. Combining this theorem with Lemma 5.2, we see that the path length for $2^{b-1} < d < 2^b$ is simply $G(2^b - d)$. Therefore, the average path length over all distances $0 \leq d < 2^b$ is $[1 + 2\sum_{d=1}^{d=2^b-1} G(d)]/2^b$. (The additive constant 1 appears because the path length for $d = 0$ is zero, while it is 1 for $d = 2^{b-1}$.) It turns out that the computation of this average is greatly simplified if we analyze the automaton in Figure 2.

We have already seen that the path length for a given distance d is just the number of thick edges traversed by the automaton when processing the b-bit binary representation of d. Computing the average path length simply requires us to run this automaton

on all possible b-bit binary strings and compute the average number of thick edges traversed. This suggests an interesting approach for analysis.

We visualize the automaton as describing a 4-state Markov chain (each transition has probability 1/2) and we count the expected number of times a thick edge ($S_0 \xrightarrow{1} S_1$ or $S_2 \xrightarrow{0} S_3$) is traversed, in b steps. 1 in either d' or d'', thereby contributing to the sum $H(d')+H(d'')$. The stationary distribution for the 4-state Markov chain would be [1/3 1/6 1/3 1/6]. Assuming that the probability of being in a particular state converges to the stationary distribution very quickly (after a small number of bits have been processed by the automaton), the expected number of thick edges taken is roughly $(\frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3}) \cdot b = b/3$. Therefore, average path length should be $\approx b/3$ for large values of b. Our formal analysis validates this intuition by computing the exact probability distribution after ℓ bits are seen by the automaton, and computing the expected number of thick edges taken using exact probabilities.

For $\ell \geq 1$, let $A_{\ell,s}$ denote the fraction of binary strings of length exactly ℓ that cause the automaton to stop in state S_s on reading them. From Figure 2, it follows that for $1 \leq \ell < b$, $A_\ell = A_0 P^\ell$, where matrix P and vectors A_ℓ and A_0 are defined as follows:

$$P = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix}$$

$$A_\ell = [\ A_{\ell,0} \quad A_{\ell,1} \quad A_{\ell,2} \quad A_{\ell,3}\ ]$$
$$A_0 = [\ 1 \quad 0 \quad 0 \quad 0\ ]$$

LEMMA 5.3. *For $1 \leq \ell < b$,*

$$A_{\ell,0} = \lceil 2^\ell/3 \rfloor / 2^\ell \qquad A_{\ell,1} = \lceil 2^\ell/6 \rfloor / 2^\ell$$
$$A_{\ell,2} = \lfloor 2^\ell/3 \rfloor / 2^\ell \qquad A_{\ell,3} = \lfloor 2^\ell/6 \rfloor / 2^\ell$$

Proof. By induction. The base case ($\ell = 1$) is true because $A_1 = [\ 1/2 \quad 1/2 \quad 0 \quad 0\ ]$. The induction step follows from the fact that for $1 \leq \ell < b$,

$$A_{\ell+1,0} = [A_{\ell,0} + A_{\ell,1} + A_{\ell,3}]/2 \qquad A_{\ell+1,1} = A_{\ell,0}/2$$
$$A_{\ell+1,2} = [A_{\ell,1} + A_{\ell,2} + A_{\ell,3}]/2 \qquad A_{\ell+1,3} = A_{\ell,2}/2$$

□

THEOREM 5.3. *The average path length for both* LEFT-TO-RIGHT CHAINING *and* RIGHT-TO-LEFT BIDIRECTIONAL GREEDY *is $b/3 + \Theta(1)$.*

Proof. The average number of thick lines encountered for strings of length b is given by $\sum_{\ell=0}^{b-1} [A_{\ell,0}/2 + A_{\ell,2}/2]$. Plugging probabilities from Lemma 5.3 and using the identity $\lceil 2^\ell/3 \rceil + \lfloor 2^\ell/3 \rfloor = 2^{\ell+1}/3 + (-1)^\ell/3$ for $\ell \geq 0$, the sum simplifies to $\frac{b}{3} + \frac{1}{9}(1 - (\frac{-1}{2})^b)$. □

6 Link Congestion

We now study the fraction of times each edge in the graph is used, in each direction, in order to perform routing from every node to every other node. We visualize an undirected edge between x and y as consisting of two *links*, (x, y) and (y, x), in order to model the connection of real computers on the Internet.

The discussion so far did not freeze the exact sequence of steps used in order to route a given distance d. In the following discussion, we will still not freeze this sequence, but will require that all nodes use the same sequence of steps in routing for each distance d. Let us define the fraction of times a link is followed from node x to node y in the routing paths from every node to every other node to be the *congestion* of link (x, y), $C(x, y)$. For notational convenience, let us assume that all arithmetic on node identifiers is modulo 2^b.

Let $N(a, (x, y))$ represent the number of times a link (x, y) is used by all the routes from a to every other node. Consider a link (x, y) where $y - x$ is 2^k. The congestion $C(x, y)$ is given by $2^{2b}C(x, y)$ $= \sum_{a=0}^{2^b-1} N(a, (x, y))$ $= \sum_{a=0}^{2^b-1} N(0, (x - a, y - a))$ $= \sum_{p=0}^{2^b-1} N(0, (p, p + 2^k))$. This quantity is simply the number of times a clockwise link of length 2^k is used in routing from node 0 to every other node. (Also note that this implies that for a given k, the congestion on link $(x, x + 2^k)$ is the same for all x.) By a similar argument, we see that all anti-clockwise links of the same length are also used an equal number of times.

THEOREM 6.1. *For* CHOICE OF TWO *routing, and for odd b[2], link congestion on all links is $C = \frac{2^{-b}}{4}(1 - \sqrt{1/8\pi b} + o(1))$, except on diameter links where it is $2C$, and on anti-clockwise links of length 1 where it is $C + 2^{-(b+1)}$.*

Proof. Observe that the average number of links used on a clockwise route from node 0 is $b/4 - \sqrt{b/8\pi} + \Theta(1)$ for odd b. Since links of all lengths are equally likely to be used, link congestion of all clockwise links is $\frac{2^{-b}}{4}(1 - \sqrt{1/8\pi b} + o(1))$. The same argument applies to anti-clockwise links. Since there are only half as many diameter links as there are links of other lengths, the congestion on them is twice that on other links. Observe that a backward link of length 1 is used on every anti-clockwise route, leading to an additional congestion of $2^{-(b+1)}$ on them. □

LEMMA 6.1. *Link congestion is symmetric for* RIGHT-TO-LEFT CHAINING, *i.e., $C(x, y) = C(y, x)$.*

[2]For even b, the congestion on clockwise links is $C' = 2^{-(b+2)}$ while that on anti-clockwise links is $2C - C'$. The same exceptions apply for unit-length backward links and diameter links.

Proof. This follows from a symmetry argument; link (x, y) is used by a route from p to q iff link (y, x) is used by the route from $(x + y - p)$ to $(x + y - q)$. Summing over all such p and q leads us to conclude that $C(x, y) = C(y, x)$. □

THEOREM 6.2. *For* RIGHT-TO-LEFT CHAINING*, link congestion on a link of length* 2^k *is* $C = \frac{2^{-b}}{6}(1 + \frac{(-1)^k}{2^{k+1}})$ *except on diameter links where it is* $2C$*.*

Proof. The fraction of times edges of length 2^k are used in routing from a single node is simply the probability of making a state transition along a thick edge, when seeing the $(k+1)^{th}$ input bit. This probability is given by $\frac{1}{2}(A_k[0] + A_k[2])$. Dividing this quantity by the number of links of length 2^k, we get the desired result. Observe that there are 2^{b+1} links of each length 2^k, except for the case $k = b - 1$ when there are only 2^b links. □

7 Chord in Base-k

In this section, we study a natural generalization of Chord, similar to higher-base hypercube topologies. Consider k^b nodes arranged clockwise in a circle with labels going from 0 to $k^b - 1$. An edge connects a pair of nodes iff they are separated by a clockwise or anti-clockwise distance of αk^β for some $0 < \alpha < b$ and $0 \leq \beta < b$. Chord in base-k is of considerable practical interest because it offers routing in fewer steps than base-2 Chord when $k > 2$.

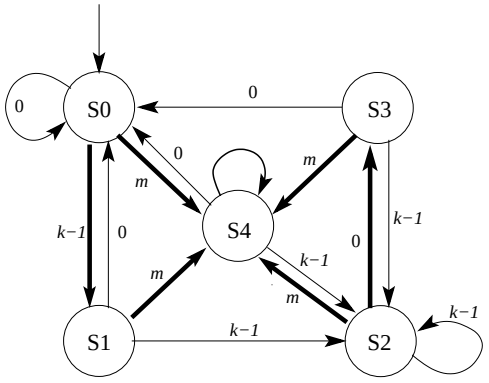

Figure 4: State machine diagram for algorithm RIGHT-TO-LEFT CHAINING FOR BASE-k.

RIGHT-TO-LEFT CHAINING FOR BASE-k: The algorithm uses the automaton in Figure 4, which generalizes the one in Figure 2. An extra state has been introduced to accommodate the middle digits, i.e., the digits other than 0 and $k - 1$. State S_4 can actually be merged with state S_3 to obtain a smaller automaton. However, we keep the two states separate for clarity.

The input string d comprises of digits from the set $\{0, 1, \ldots k - 1\}$ and is scanned from `right` to `left`. Symbol m is a short-hand for the "middle digits", i.e., the set $\{1, 2, \ldots k - 2\}$. Each transition also produces a pair of digits as output, the first digit for d' and the second for d''. The start state is S_0. All thin edges produce $\langle 0, 0\rangle$ as output. All thick edges produce non-zero output and require *lookahead*:

$S_0 \overset{k-1}{\rightarrow} S_1$: `if` (next digit is $k-1$) `then output` $\langle 0, 1\rangle$ `else output` $\langle k-1, 0\rangle$

$S_2 \overset{0}{\rightarrow} S_3$: `if` (next digit is $k-1$) `then output` $\langle 0, k-1\rangle$ `else output` $\langle 1, 0\rangle$

$X \overset{m}{\rightarrow} S_4$: `if` (next digit is $k-1$) `then output` $\langle 0, m-1\rangle$ `else output` $\langle m, 0\rangle$

where X denotes any of the five states.

THEOREM 7.1. *The average length of shortest paths for base-k Chord on* k^b *nodes is* $\frac{k-1}{k+1}b + \Theta(1)$*.*

Proof. The stationary distribution for the Markov chain associated with the automaton in Figure 4 is $[\ \frac{1}{k+1}\ \ \frac{1}{k(k+1)}\ \ \frac{1}{k+1}\ \ \frac{k-1}{k(k+1)}\ \ \frac{k-2}{k+1}\ ]$. A cost of 1 hop is paid every time a thick edge is traversed. Continuing in the same fashion as in the proof of Theorem 5.3, it can be shown that the average number of thick edge traversals is $\frac{k-1}{k+1}b + \Theta(1)$. □

8 The HyperSkewbe

We define the HyperSkewbe, a skew-symmetric variant of the Hypercube, as follows:

- The HyperSkewbe of dimension 1, S_1 is a graph consisting of two nodes, labeled 0 and 1, with an edge between the two nodes.
- (Recurrence) Consider one copy of S_k, and suffix all node identifiers with a 0 to create a graph $S_{k,0}$. Consider another copy of S_k and suffix all node identifiers with a 1 to create $S_{k,1}$. Now, create an edge between node x in $S_{k,0}$ and node $(x - 1) \bmod 2^{k+1}$ in $S_{k,1}$. Call the resulting graph S_{k+1}.

We observe that the above recurrence is almost identical to the standard hypercube definition. The only difference is that node x in $S_{k,0}$ is attached to $(x - 1) \bmod 2^{k+1}$ to create the HyperSkewbe, instead of to $x + 1$, which would have resulted in the standard Hypercube. This small variation in the definition results in an intriguing structure which appears to have

properties rather different from that of the Hypercube. For example, the diameter of the k-dimensional Hyper-Skewbe is smaller than k for all $k > 2$.

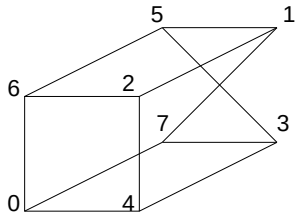

Figure 5: 3-dimensional Hyperskewbe on 8 nodes.

Figure 5 shows a three-dimensional HyperSkewbe. The Chord network is simply the union of the hypercube of dimension k with the HyperSkewbe of dimension k, and it appears that it is the presence of this Hyper-Skewbe that makes the routing properties of Chord so different from that of the standard hypercube.

Let us now consider routing on the HyperSkewbe. One algorithm for routing is to perform right-to-left bit-fixing, which results in an average path length of $b/2$ on the b-dimensional HyperSkewbe. One improvement to this routing algorithm is the following: Suppose we wish to route from node x to node y. We find the length of the route obtained by right-to-left bit-fixing to convert x to y. We also find the length of the route obtained by right-to-left bit-fixing converting y to x. In general, these two routes, and their lengths, are not identical. We can then choose the shorter of these two routes as the route from x to y. Experiments indicate that this idea improves upon simple right-to-left bit-fixing considerably. However, this algorithm is not optimal either. The characterization of optimal routes on the HyperSkewbe appears to be an interesting open problem.

References

[1] I. Stoica, R. Morris, D. Liben-Lowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications*, IEEE/ACM Transactions on Networking, Vol. 11, No. 1 (2003), pp. 17–32.

[2] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation* (2nd Ed.), Pearson Addison Wesley, (2000), pp. 1-521.