Network computing capacity for the reverse butterfly network

Rathinakumar Appuswamy, Massimo Franceschetti,
Nikhil Karamchandani, and Kenneth Zeger

***Abstract*—We study the computation of the arithmetic sum of the q-ary source messages in the reverse butterfly network. Specifically, we characterize the maximum rate at which the message sum can be computed at the receiver and demonstrate that linear coding is suboptimal.**

I. Introduction

Suppose a set of source nodes in a network generates independent messages and a single receiver node computes a function of these messages. The objective is to characterize the maximum rate of computation, that is the maximum number of times the specified function can be computed per network usage. In [2], we presented a network computation model that is closely related to network coding [1], [13], thus extending network "coding" to network "computing".

Network coding with a single receiver is equivalent to the special case of network computing when the function to be computed is the (vector) identity map - that is, the receiver wants to reproduce all of the messages generated by the sources. Routing is known to be rate-optimal for single receiver network coding [9]. We extend the network coding model to computation of arbitrary functions of source messages. Our work is related to that of Ramamoorthy [11], who studied computing the parity of a collection of binary sources in a network with two sources and arbitrary number of receivers, or vice versa; however, he considered only the existence of a solution, rather than the rate at which the solution can be computed. Problems related to function computation have been studied in a such areas as communication complexity [8], [12], average consensus [4], [6], and distributed computation [3], [10]. The reader is referred to [7] for a review of various approaches to the problem. The model we use is concerned with the maximum possible rate of computing a given function at a node where, the set of allowable interactions is constrained by the network topology.

In this paper, we study the problem of computing the arithmetic sum of the source messages in the reverse butterfly network (obtained by reversing the direction of all the edges in the well known multicast butterfly network) as an illustrative example. We first find the maximum rate at which a modulo sum can be computed and use that result to determine the maximum possible rate of computation of the arithmetic sum (i.e. over $\mathbf{Z}$). We then compare the maximum achievable computing rate to that using only linear coding. These results naturally generalize to arbitrary acyclic networks having a single receiver and will be presented in a future publication.

This work was supported by the National Science Foundation, AFOSR, and the UCSD Center for Wireless Communications

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407. Emails: rathnam@ucsd.edu, massimo@ece.ucsd.edu, nikhil@ucsd.edu, zeger@ucsd.edu

This paper appeared at the the IEEE International Symposium on Information Theory (ISIT), in Seoul, Korea, June 28 – July 3, 2009.

A. Network model and preliminaries

In this paper, a *network* $\mathcal{N}$ for computation consists of a finite, directed acyclic multigraph $G = (\mathcal{V}, \mathcal{E})$, a set of *source nodes* $\mathcal{S} \subseteq \mathcal{V}$, and a single *receiver* $T \in \mathcal{V} - \mathcal{S}$. Throughout, let $s = |\mathcal{S}|$. Such a network is denoted by $\mathcal{N} = (G, \mathcal{S}, T)$. We will assume (without loss of generality) that if a node has no in-edges, then it is a source node. An *alphabet* is a finite set of size at least two. Each source generates *messages*, which are symbols from a fixed alphabet[1] $\mathcal{A}$. The objective of the receiver is to compute a certain function of these messages.

Let $\mathcal{B}$ be an arbitrary alphabet. A *target function* is any map of the form

$$f : \mathcal{A}^s \to \mathcal{B}.$$

Let $+$ denote the arithmetic sum operation (i.e. ordinary addition) and let $\oplus$ denote modulo addition (for a specified modulus). Some example target functions are defined below.

Example I.1. Let $\mathcal{B} = \mathcal{A}^s$. The *identity target function* is

$$f(x_1, \ldots, x_s) = (x_1, \ldots, x_s).$$

Example I.2. Let $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and $\mathcal{B} = \{0, 1, \ldots, (q-1)s\}$ with $q \geq 2$. The *arithmetic sum target function* is

$$f(x_1, \ldots, x_s) = \sum_{i=1}^{s} x_i.$$

Example I.3. Let $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and $\mathcal{B} = \{0, 1, \ldots, r-1\}$. For $q, r \geq 2$, the *mod r sum target function* is

$$f(x_1, \ldots, x_s) = x_1 \oplus x_2 \oplus \ldots \oplus x_s.$$

The network computation problem consists of a single-receiver network $\mathcal{N}$, and a target function f, whose arguments are the network source messages. The goal is to compute f at the receiver T.

We will view each network source node in $\mathcal{S}$ as generating a vector of k alphabet symbols (e.g. modeling a source output over k consecutive time units). Every out-edge of each node in $\mathcal{V}$ carries a vector of n alphabet symbols, which is a function of

[1]For simplicity we assume each source has associated with it exactly one message, but all of the results in this paper can readily be extended to the more general case.

the vectors carried by the in-edges to the node and the node's message vector if it is a source. The objective of the receiver is to construct a vector of k alphabet symbols, such that for each $i \in \{1, 2, \ldots, k\}$, the i-th component of the receiver's computed vector equals the value of the target function f applied to the i-th components of the source message vectors.

Let $\mathcal{S} = \{\mu_1, \ldots, \mu_s\}$ and fix a mapping

$$\alpha \ : \ \mathcal{S} \to \mathcal{A}^k.$$

For each $m \in \{1, 2, \ldots, s\}$, we say that the k-dimensional quantity $\alpha(\mu_m)$ is a *message vector* that is *generated* by the source μ_m. The i-th component of $\alpha(\mu_m)$ is denoted by $\alpha(\mu_m)_i$.

For each node u, let $\mathcal{E}_i(u)$ denote the set of in-edges of u. For each network edge $e = (u, v) \in \mathcal{E}$, an *encoding function* g_e is a mapping

$$g_e \ : \begin{cases} \mathcal{A}^{n|\mathcal{E}_i(u)|} \times \mathcal{A}^k \ \to \ \mathcal{A}^n & \text{if } u \in \mathcal{S} \\ \mathcal{A}^{n|\mathcal{E}_i(u)|} \ \to \ \mathcal{A}^n & \text{otherwise.} \end{cases}$$

A *decoding function* ψ (at the receiver T) is a mapping

$$\psi \ : \ \mathcal{A}^{n|\mathcal{E}_i(T)|} \ \to \ \mathcal{B}^k.$$

A (k, n) *network code* (with respect to a particular alphabet $\mathcal{A}$) for a network is a collection of encoding functions, one for each network edge, together with a decoding function at the receiver. For each edge e, let $z_e \in \mathcal{A}^n$ denote the vector carried by e and denote the in-edges of the receiver by $e_1, e_2, \ldots, e_{|\mathcal{E}_i(T)|}$. A (k, n) network code is called *a solution for computing f* if, for each $i \in \{1, 2, \ldots, k\}$, and for each mapping α, we have

$$\psi\left(z_{e_1}, \cdots, z_{e_{|\mathcal{E}_i(T)|}}\right)_i = f\left(\alpha(\mu_1)_i, \cdots, \alpha(\mu_s)_i\right).$$

In this case, we say the rational number k/n is an *achievable computing rate*. When the alphabet $\mathcal{A}$ is a ring, a (k, n) network code is said to be *linear* if all the encoding functions are linear (i.e., at each node, its out-edges carry a linear combination with matrix coefficients over $\mathcal{A}$ of all the vectors on the in-edges of that node). We do not require the decoding function ψ to be linear in a linear network code, thus allowing a linear network code to compute non-linear target functions.

The *computing capacity* of a network $\mathcal{N}$ with respect to target function f is

$$\mathcal{C}(\mathcal{N}, f) \ = \ \sup\Big\{\frac{k}{n} \ : \ \exists (k, n) \text{ network coding solution for computing } f\Big\}.$$

The computing capacity is thus the supremum of all achievable computing rates for a network. The *linear computing capacity* $\mathcal{C}_{\text{lin}}(\mathcal{N}, f)$ and *routing computing capacity* are defined similarly by restricting the set of allowable encoding functions.

II. Summary of the results

Figure 1 shows the well-known multicast butterfly network (introduced in [1]). The network shown in Figure 2 is obtained by reversing the direction of all the edges of the butterfly network, and we call the result the *reverse butterfly network*

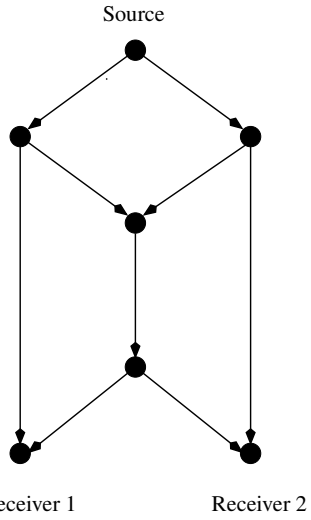

Fig. 1. The butterfly network.

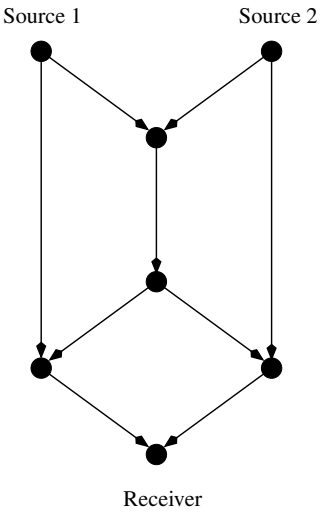

Fig. 2. The reverse butterfly network.

The reverse butterfly network has sources $\mathcal{S} = \{\mu_1, \mu_2\}$ and receiver node T.

Theorem II.1. *Let $\mathcal{N}$ denote the reverse butterfly network and let the alphabet be $\mathcal{A} = \{0, 1, \ldots, q-1\}$. If f is the arithmetic sum target function, then the computing capacity of $\mathcal{N}$ is*

$$\mathcal{C}(\mathcal{N}, f) = \frac{2}{\log_q(2q-1)}.$$

Theorem II.2. *Let $\mathcal{N}$ denote the reverse butterfly network and let the alphabet be $\mathcal{A} = \{0, 1, \ldots, q-1\}$. If f is the arithmetic sum target function, then any linear computing capacity of $\mathcal{N}$ is*

$$\mathcal{C}_{lin}(\mathcal{N}, f) = 1.$$

The phrase "any linear computing capacity" in Theorem II.2 refers to the possibility that different underlying ring structures of the alphabet $\mathcal{A}$ can give rise to different types of linearity.

Remark II.3. *The arithmetic sum can be computed in the reverse butterfly network at a computing rate of* 1 *using only*

routing (by sending $\alpha(\mu_1)$ down the left side and $\alpha(\mu_2)$ down the right side of the graph). From a simple cutset argument, it follows that the routing computing capacity is equal to 1 for all $q \geq 2$.

Remark II.4. *The computing capacity $\mathcal{C}(\mathcal{N}, f)$ obtained in Theorem II.1 depends on the coding alphabet $\mathcal{A}$. In contrast, for ordinary network coding (i.e. when the target function is the identity map), the coding capacity and routing capacity are known to be independent of the coding alphabet used [5].*

Remark II.5. *The ratio of the coding capacity to the routing capacity for the multicast butterfly network with two messages is $4/3$ (see [5]), i.e. coding provides an advantage of about 33%. The corresponding ratio of computing rates for the reverse butterfly network increases as a function of q from approximately 1.26 (i.e. 26%) to 2 (i.e. 100%) as $q \to \infty$. Furthermore, in contrast to the multicast butterfly network, where the coding capacity is equal to the linear coding capacity, in the reverse butterfly network the nonlinear computing capacity is strictly greater than any linear computing capacity.*

Remark II.6. *It was pointed out in [5], that an open question is whether the coding capacity of a network can be irrational. Our Theorem II.1 demonstrates that the computing capacity of a network can be irrational.*

A. Discussion

In the next section, the proof of achievability in Theorem II.1 will use a result on computing the modulo sum in the reverse butterfly network. It is based on the observation that computing the arithmetic sum of the source messages over an alphabet $\mathcal{A} = \{0, 1, \ldots, q\}$ can be accomplished by computing the modulo sum of the sources for sufficiently large modulus.

III. Proofs

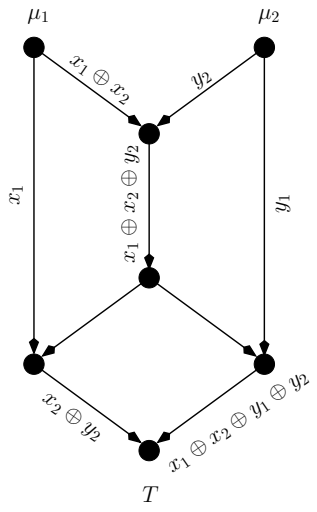

Fig. 3. The reverse butterfly network with a solution for computing the $\bmod q$ sum target function. The messages vectors are $\alpha(\mu_1) = (x_1, x_2)$, $\alpha(\mu_2) = (y_1, y_2)$, and T computes $(x_1 \oplus y_1, x_2 \oplus y_2)$. '$\oplus$' denotes $\bmod q$ addition.

Lemma III.1. *Let $\mathcal{N}$ denote the reverse butterfly network and let the alphabet be $\mathcal{A} = \{0, 1, \ldots, q-1\}$. If f denotes the $\bmod q$ sum target function, then the computing capacity of $\mathcal{N}$ is*

$$\mathcal{C}(\mathcal{N}, f) = 2.$$

Proof: We have $\mathcal{C}(\mathcal{N}, f) \leq 2$ from [2, Theorem II.1]. To establish the lower bound, let $k = 2$ and $n = 1$. Consider the linear code shown in Figure 3, where '$\oplus$' indicates the $\bmod q$ sum and where μ_1 emits the vector (x_1, x_2) and μ_2 emits the vector (y_1, y_2). The receiver node T gets $x_1 \oplus y_1$ and $x_1 \oplus x_2 \oplus y_1 \oplus y_2$ on its in-edges, from which it can compute $x_2 \oplus y_2$ by subtraction. This code achieves a computing rate of 2. ∎

A. Proof of Theorem II.1

Proof: We have $\mathcal{C}(\mathcal{N}, f) \leq 2/\log_q(2q-1)$ from [2, Theorem II.1]. To establish the lower bound, we use the fact the that arithmetic sum of two elements from $\mathcal{A} = \{0, 1, \ldots, q-1\}$ is equal to their $\bmod 2q-1$ sum. Let the reverse butterfly network have alphabet $\hat{\mathcal{A}} = \{0, 1, \ldots, 2(q-1)\}$. From Lemma III.1 (with alphabet $\hat{\mathcal{A}}$), the $\bmod 2q-1$ sum can be computed in $\mathcal{N}$ at rate 2. Indeed for every $n \geq 1$, there exists a $(2n, n)$ network code for computing at rate 2 the $\bmod 2q-1$ sum. So for the remainder of this proof, let $k = 2n$. Furthermore, every such code using $\hat{\mathcal{A}}$ can be "simulated" using $\mathcal{A}$ by a corresponding $(2n, \lceil n \log_q(2q-1) \rceil)$ code for computing the $\bmod 2q-1$ sum, as follows. Let n' be the smallest integer such that $q^{n'} \geq (2q-1)^n$, i.e., $n' = \lceil n \log_q(2q-1) \rceil$. Let $g : \hat{\mathcal{A}}^n \to \mathcal{A}^{n'}$ be an injection (which exists since $q^{n'} \geq (2q-1)^n$) and let the function g^{-1} denote the inverse of g on it's image $g(\hat{\mathcal{A}})$. Let $x^{(1)}, x^{(2)}$ denote the first and last, respectively, halves of the message vector $\alpha(\mu_1) \in \mathcal{A}^{2n}$, where we view $x^{(1)}$ and $x^{(2)}$ as lying in $\hat{\mathcal{A}}^n$ (since $\mathcal{A} \subseteq \hat{\mathcal{A}}$). The corresponding vectors $y^{(1)}, y^{(2)}$ for the source μ_2 are similarly defined.

Figure 4 illustrates a $(2n, n')$ code for network $\mathcal{N}$ using alphabet $\mathcal{A}$ where, '$\oplus$' denotes the $\bmod 2q-1$ sum. Each of the nodes in $\mathcal{N}$ converts each of the received vectors over $\mathcal{A}$ into a vector over $\hat{\mathcal{A}}$ using the function g^{-1}, then performs the node function in Figure 3 over $\hat{\mathcal{A}}$, and finally converts the result back to $\mathcal{A}$. Similarly, the receiver node T computes the component-wise arithmetic sum of the source message vectors $\alpha(\mu_1)$ and $\alpha(\mu_2)$ as

$$\begin{aligned}
&\alpha(\mu_1) + \alpha(\mu_2) \\
&= \big(g^{-1}(g(x^{(1)} \oplus x^{(2)} \oplus y^{(1)} \oplus y^{(2)})) \ominus g^{-1}(g(x^{(2)} \oplus y^{(2)})), \\
&\qquad g^{-1}(g(x^{(2)} \oplus y^{(2)}))\big) \\
&= (x^{(1)} \oplus y^{(1)}, x^{(2)} \oplus y^{(2)}).
\end{aligned}$$

For any $n \geq 1$, the above solution computes the arithmetic sum target function in $\mathcal{N}$ at a rate of

$$\frac{k}{n'} = \frac{2n}{\lceil n \log_q(2q-1) \rceil}.$$

Thus for any $\epsilon > 0$, by choosing n large enough we have a

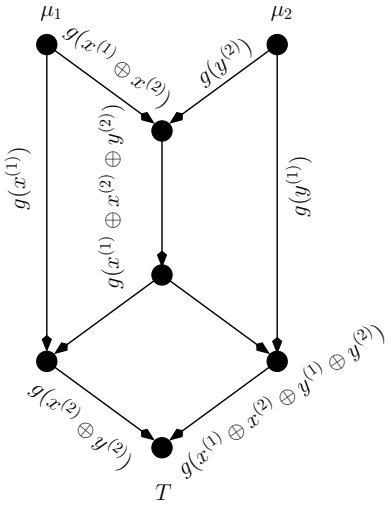

Fig. 4. The reverse butterfly network with a solution for computing the arithmetic sum target function when $\mathcal{A} = \{0, 1, \ldots, q-1\}$. '$\oplus$' denotes $\bmod\ 2q-1$ addition.

scheme which achieves a computing rate of at least

$$\frac{2}{\log_q(2q-1)} - \epsilon.$$

■

B. Proof of Theorem II.2

Proof: The lower bound follows from the fact that routing achieves a computing rate of 1.

Let e_1, e_2 denote the two in-edges of the receiver T. For any (k, n) linear network solution, the length-n vectors on edges e_1 and e_2 can be written as[2] $M_1\alpha(\mu_1) + M_2\alpha(\mu_2)$ and $M_3\alpha(\mu_1) + M_4\alpha(\mu_2)$, respectively, where M_1, M_2, M_3, and M_4 are $n \times k$ matrices over $\mathcal{A}$.

We will show that distinct pairs $(\alpha(\mu_1), \alpha(\mu_2))$ of message vectors lead to distinct pairs of values carried on e_1 and e_2, and thus $k \leq n$. Suppose α and α' are mappings such that message vectors $(\alpha(\mu_1), \alpha(\mu_2)) = (w^{(1)}, w^{(2)})$ and $(\alpha'(\mu_1), \alpha'(\mu_2)) = (w^{(1)'}, w^{(2)'})$ in $\mathcal{A}^k \times \mathcal{A}^k$ satisfy

$$(M_1w^{(1)} + M_2w^{(2)}, M_3w^{(1)} + M_4w^{(2)}) =$$
$$(M_1w^{(1)'} + M_2w^{(2)'}, M_3w^{(1)'} + M_4w^{(2)'}).$$

Then we have

$$M_1(w^{(1)} - w^{(1)'}) + M_2(w^{(2)} - w^{(2)'}) = \mathbf{0}$$
$$M_3(w^{(1)} - w^{(1)'}) + M_4(w^{(2)} - w^{(2)'}) = \mathbf{0}$$

where $\mathbf{0}$ denotes the length-n zero vector. That is, when the source messages are $\alpha(\mu_1) = w^{(1)} - w^{(1)'}$ and $\alpha(\mu_2) = w^{(2)} - w^{(2)'}$, the edges e_1 and e_2 each carry the all-zero vector.

On the other hand, when the source message map α is such that $\alpha(\mu_1) = \alpha(\mu_2) = \mathbf{0}$, the edges e_1 and e_2 also each carry the all-zero vector. In this case the arithmetic sum target function evaluates to $\mathbf{0} + \mathbf{0} = \mathbf{0}$, so in order to prevent ambiguity the receiver must always decode the target function value as $\mathbf{0}$ whenever $z_{e_1} = z_{e_2} = \mathbf{0}$. Since the arithmetic sum of any vectors over $\mathcal{A}$ is $\mathbf{0}$ if and only if each of the sources vectors is $\mathbf{0}$, the vectors carried on e_1 and e_2 are both zero if and only if both source vectors are zero. Thus, we must have $(w^{(1)} - w^{(1)'}, w^{(2)} - w^{(2)'}) = (\mathbf{0}, \mathbf{0})$ and therefore $w^{(1)} = w^{(1)'}$ and $w^{(2)} = w^{(2)'}$. Thus for any linear code that computes the arithmetic sum target function, every distinct pair of message vectors $(\alpha(\mu_1), \alpha(\mu_2))$ produces a distinct edge vector pair $(M_1w^{(1)} + M_2w^{(2)}, M_3w^{(1)} + M_4w^{(2)})$. Hence $|\mathcal{A}|^{2n} \geq |\mathcal{A}|^{2k}$ and the computing rate satisfies $k/n \leq 1$. ■

[2]In this proof, matrix multiplication and vector addition are performed over the ring in which the network code is linear and $\mathbf{0}$ denotes the zero vector over the same ring.

C. Conclusion

We have illustrated concepts of network computing with the reverse butterfly network by deriving its non-linear, linear, and routing computing capacities. More extensive and general results will be presented in a future publication. In particular, we can obtain the computing capacity of arbitrary single-receiver directed, acyclic networks if the target function is linear over a finite field alphabet. This result, then, can be used to provide a lower bound on the rate at which the arithmetic sum target function can be computed in such networks.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow", *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[2] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing", *Proceedings of the forty-sixth Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sept. 2008.

[3] O. Ayaso, D. Shah, and M. Dahleh, "Lower Bounds on Information Rates for Distributed Computation via Noisy Channels", *Proceedings of the forty-fifth Allerton Conference on Computation, Communication and Control*, Monticello, IL, Sept. 2007.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized Gossip Algorithms", *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, June 2008.

[5] J. Cannons, R. Dougherty, C. Freiling, and K. Zeger, "Network routing capacity", *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 777–788, March 2006.

[6] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic gossip: efficient aggregation for sensor networks", *Proceedings of the fifth international conference on Information processing in sensor networks*, Nashville, TN, April 2006, pp. 69–76.

[7] A. Giridhar and P. R. Kumar, "Toward a theory of in-network computation in wireless sensor networks", *IEEE Communications Magazine*, vol. 44, no. 4, pp. 98–107, April 2006.

[8] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, 1997.

[9] A. Rasala Lehman and E. Lehman, "Complexity classification of network information flow problems", *Proceedings of the fifteenth annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, Jan. 2004, pp. 142–150.

[10] D. Mosk-Aoyama and D. Shah, "Fast Distributed Algorithms for Computing Separable Functions", *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2997–3007, July 2008.

[11] A. Ramamoorthy, "Communicating the sum of sources over a network", *Proceedings of the IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008, pp. 1646–1650.

[12] A. C. Yao, "Some Complexity Questions Related to Distributed Computing", *Proc. of eleventh ACM Symposium on Theory of Computing*, Atlanta, GA, April 1979, pp. 209–213.

[13] R. W. Yeung, *A First Course in Information Theory*, Springer, 2002.