On Row-by-Row Coding for 2-D Constraints

Ido Tal, *Member, IEEE*, Tuvi Etzion, *Fellow, IEEE*, and Ron M. Roth, *Fellow, IEEE*

***Abstract*—A constant-rate encoder–decoder pair is presented for a fairly large family of two-dimensional (2-D) constraints. Encoding and decoding is done in a row-by-row manner, and is sliding-block decodable.**

Essentially, the 2-D constraint is turned into a set of independent and relatively simple one-dimensional (1-D) constraints; this is done by dividing the array into fixed-width vertical strips. Each row in the strip is seen as a symbol, and a graph presentation of the respective 1-D constraint is constructed. The maxentropic stationary Markov chain on this graph is next considered: a perturbed version of the corresponding probability distribution on the edges of the graph is used in order to build an encoder which operates *in parallel* on the strips. This perturbation is found by means of a network flow, with upper and lower bounds on the flow through the edges.

A key part of the encoder is an enumerative coder for constant-weight binary words. A fast realization of this coder is shown, using floating-point arithmetic.

***Index Terms*—Enumerative coding, flow networks, Kronecker product, parallel encoding, row-by-row coding, runlength-limited (RLL) constraints, two-dimensional (2–D) constraints.**

I. Introduction

LET $G = (V, E, L)$ be an edge-labeled directed graph (referred to hereafter simply as a graph), where V is the vertex set, E is the edge set, and $L : E \to \Sigma$ is the edge labeling taking values on a finite alphabet Σ [1, Sec. 2.1]. We require that the labeling L is deterministic: edges that start at the same vertex have distinct labels. We further assume that G has finite memory [1, Sec. 2.2.3]. The one-dimensional (1-D) *constraint* $S = S(G)$ that is presented by G is defined as the set of all words that are generated by paths in G (i.e., the words are obtained by reading-off the edge labels of such paths). Examples of 1-D constraints include runlength-limited (RLL) constraints [1, Sec. 1.1.1], symmetric RLL (SRLL) constraints [2], and the charge constraints [1, Sec. 1.1.2].

Manuscript received August 03, 2008. Current version published July 15, 2009. The work of T. Etzion was supported in part by the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel, under Grant 2006097. The work of R. M. Roth was supported in part by the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel, under Grant 2002197. The material in this paper was presented in part at the IEEE International Symposium on Information Theory (ISIT), Seattle, WA, July 2006. This work was done while I. Tal was with the Computer Science Department, Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel.

I. Tal is with the Center for Magnetic Recording Research (CMMR), University of California, San Diego, La Jolla, CA 92093-0401 USA (e-mail: idotal@ieee.org).

T. Etzion and R. M. Roth are with the Computer Science Department, Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel (e-mail: etzion@cs.technion.ac.il; ronny@cs.technion.ac.il).

Communicated by L. M. G. M. Tolhuizen, Associate Editor for Coding Theory.

Color versions of Figures 1 and 11 in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2009.2023754

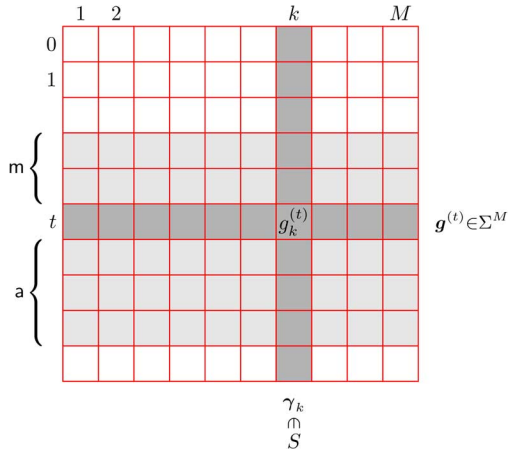


Fig. 1. Array corresponding to an M-track parallel encoder.

The capacity of S is given by

$$\mathsf{cap}(S) = \lim_{\ell \to \infty} (1/\ell) \cdot \log_2 \left| S \cap \Sigma^\ell \right|.$$

An M-track *parallel encoder* for $S = S(G)$ at rate R is defined as follows (see Fig. 1).

1) At stage $t = 0, 1, 2, \ldots$, the encoder (which may be state-dependent) receives as input $M \cdot R$ (unconstrained) information bits.
2) The output of the encoder at stage t is a word $\boldsymbol{g}^{(t)} = (g_k^{(t)})_{k=1}^M$ of length M over Σ.
3) For $1 \le k \le M$, the kth track $\boldsymbol{\gamma}_k = (g_k^{(t)})_{t=0}^{\ell-1}$ of any given length ℓ, belongs to S.
4) There are integers $\mathsf{m}, \mathsf{a} \ge 0$ such that the encoder is (m, a)-*sliding-block decodable* (in short, (m, a)-SBD): for $t \ge \mathsf{m}$, the $M \cdot R$ information bits which were input at stage t are uniquely determined by (and can be efficiently calculated from) $\boldsymbol{g}^{(t-\mathsf{m})}, \boldsymbol{g}^{(t-\mathsf{m}+1)}, \ldots, \boldsymbol{g}^{(t+\mathsf{a})}$.

The decoding window size of the encoder is $\mathsf{m} + \mathsf{a} + 1$, and it is desirable to have a small window to avoid error propagation. In this work, we will be mainly focusing on the case where $\mathsf{a} = 0$, in which case the decoding requires no look-ahead.

In [3], it was shown that by introducing parallelism, one can reduce the window size, compared to conventional serial encoding. Furthermore, it was shown that as M tends to infinity, there are $(0, 0)$-SBD parallel encoders whose rates approach $\mathsf{cap}(S(G))$. A key step in [3] is using some perturbation of the conditional probability distribution on the edges of G, corresponding to the maxentropic stationary Markov chain on G. However, it is not clear how this perturbation should be done: a naive method will only work for unrealistically large M. Also, the proof in [3] of the $(0, 0)$-SBD property is only probabilistic and does not suggest encoders and decoders that have an acceptable running time.

In this work, we aim at making the results of [3] more tractable. At the expense of possibly increasing the memory of the encoder (up to the memory of G) we are able to define a suitable perturbed distribution explicitly, and provide an efficient algorithm for computing it. Furthermore, the encoding and decoding can be carried out in time complexity $O(M \log^2 M \log\log M)$, where the multiplying constants in the $O(\cdot)$ term are polynomially large in the parameters of G.

Denote by $\mathrm{diam}(G)$ the diameter of G (i.e., the longest shortest path between any two vertices in G) and let $A_G = (a_{i,j})$ be the adjacency matrix of G, i.e., $a_{i,j}$ is the number of edges in G that start at vertex i and terminate in vertex j. Our main result, specifying the rate of our encoder, is given in the next theorem.

Theorem 1: Let G be a deterministic graph with memory m. For M sufficiently large, one can efficiently construct an M-track $(\mathsf{m}, 0)$-SBD parallel encoder for $S = S(G)$ at a rate R such that

$$R \geq \mathsf{cap}(S(G)) \left(1 - \frac{|V| \,\mathrm{diam}(G)}{2M}\right) - O\left(\frac{|V|^2 \log(M \cdot a_{\max}/a_{\min})}{M - |V| \,\mathrm{diam}(G)/2}\right) \quad (1)$$

where $a_{\min}$ (respectively, $a_{\max}$) is the smallest (respectively, largest) *nonzero* entry in A_G.

The structure of this paper is as follows. In Section II, we show how parallel encoding can be used to construct an encoder for a two–dimensional (2-D) constraint. As we will show, a parallel encoder is essentially defined through what we term a multiplicity matrix. Section III defines how our parallel encoder works, assuming its multiplicity matrix is given. Then, in Section IV, we show how to efficiently calculate a good multiplicity matrix. Although 2-D constraints are our main motivation, Section V shows how our method can be applied to 1-D constraints. Section VI defines two methods by which the rate of our encoder can be slightly improved. Finally, in Section VII we show a method of efficiently realizing a key part of our encoding procedure.

II. Two-Dimensional Constraints

Our primary motivation for studying parallel encoding is to show an encoding algorithm for a family of 2-D constraints.

The concept of a 1-D constraint can formally be generalized to two dimensions (see [3, Sec. 1]). Examples of 2-D constraints are 2-D RLL constraints [4], 2-D SRLL constraints [2], and the kings constraint (termed the square constraint in [5]). Let $\mathbb{S}$ be a given 2-D constraint over a finite alphabet Σ. We denote by $\mathbb{S}[\ell, h]$ the set of all $\ell \times h$ arrays in $\mathbb{S}$. The capacity of $\mathbb{S}$ [6] is given by

$$\mathsf{cap}(\mathbb{S}) = \lim_{\ell,h\to\infty} \frac{1}{\ell \cdot h} \cdot \log_2 |\mathbb{S}[\ell, h]| \,.$$

Suppose we wish to encode information to an $\ell \times h$ array which must satisfy the constraint $\mathbb{S}$; namely, the array must be an element of $\mathbb{S}[\ell, h]$. As a concrete example, consider the kings constraint [5]: its elements are all the binary arrays in which no two "1" symbols are adjacent on a row, column, or diagonal.

$$\begin{array}{cccc|c|cccc|c|cccc} 0&0&1&0&0&1&0&1&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&1&0&1&0&0&1 \end{array}$$

Fig. 2. Binary array satisfying the kings constraint, partitioned into data strips of width $w = 4$ and merging strips of width $b = 1$.

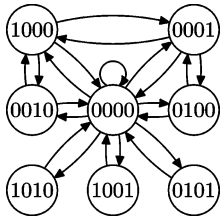


Fig. 3. Graph G whose paths generate all $\ell \times 4$ arrays satisfying the kings constraint. The label of an edge is given by the label of the vertex it enters.

We first partition our array into two alternating types of vertical strips: *data strips* having width w, and *merging strips* having width b. In our example, let $w = 4$ and $b = 1$ (see Fig. 2).

Second, we select a graph $G = (V, E, L)$ with a labeling $L : E \to \mathbb{S}[1, w]$ such that $S(G) \subseteq \mathbb{S}$, i.e., each path of length ℓ in G generates a (column) word which is in $\mathbb{S}[\ell, w]$. We then fill up the data strips of our $\ell \times h$ array with $\ell \times w$ arrays corresponding to paths of length ℓ in G. Third, we assume that the choice of b allows us to fill up the merging strips in a row-by-row (causal) manner, such that our $\ell \times h$ array is in $\mathbb{S}$. Any 2-D constraint $\mathbb{S}$ for which such w, b, and G can be found, is in the family of constraints we can code for (for example, the 2-D SRLL constraints belong to this family [2]).

Consider again the kings constraint: a graph which produces all $\ell \times w$ arrays that satisfy this constraint is given in Fig. 3. Also, for $b = 1$, we can take the merging strips to be all zero. (There are cases, such as the 2-D SRLL constraints, where determining the merging strips may be less trivial [2].)

Suppose we have an $(\mathsf{m}, 0)$-SBD parallel encoder for $S = S(G)$ at rate R with $M = (h + b)/(w + b)$ tracks. We may use this parallel encoder to encode information in a row-by-row fashion to our $\ell \times h$ array: at stage t we feed $M \cdot R$ information bits to our parallel encoder. Let $\boldsymbol{g}^{(t)} = (g_k^{(t)})_{k=1}^{M}$ be the output of the parallel encoder at stage t. We write $g_k^{(t)}$ to row t of the kth data strip, and then appropriately fill up row t of the merging strips. Decoding of a row in our array can be carried out based only on the contents of that row and the previous m rows.

Since $M \cdot R$ information bits are mapped to $M \cdot w + (M-1) \cdot b$ symbols in Σ, the rate at which we encode information to the array is

$$\frac{R}{w + b(1 - 1/M)} \leq \frac{\mathsf{cap}(S(G))}{w + b(1 - 1/M)}. \quad (2)$$

Note that if we remove the $1/M$ term (which is typically negligible) from the right-hand side of (2), we get a lower bound on $\mathsf{cap}(\mathbb{S})$, which converges to $\mathsf{cap}(\mathbb{S})$ as $w \to \infty$ (and b is kept constant). Thus, we have by Theorem 1 that the left-hand side of (2) approaches $\mathsf{cap}(\mathbb{S})$ as M and w tend to infinity. However,

there is a tradeoff: the number of vertices and edges in G will usually grow exponentially with w. Therefore, w is taken to be reasonably small. Also, recall that once the values of w and b are decided upon, the value of M is set according to the width h of the 2-D array.

Note that in our scheme, a single error generally results in the loss of information stored in the respective vertical sliding-block window. Namely, a single corrupted entry in the array may cause the loss of $\mathsf{m}+1$ rows. Thus, our method is only practical if we assume an error model in which whole rows are corrupted by errors. This is indeed the case if each row is protected by an error-correcting code (for example, by the use of unconstrained positions [7]).

III. Description of the Encoder

Let N be a positive integer which will shortly be specified. The N words $\boldsymbol{\gamma}_k = (g_k^{(t)})_{t=0}^{\ell-1}$, $1 \le k \le N$, that we will be writing to the first N tracks are all generated by paths of length ℓ in G. In what follows, we find it convenient to regard the $\ell \times N$ arrays $(\boldsymbol{\gamma}_k)_{k=1}^{N} = (g_k^{(t)})_{t=1}^{\ell}{}_{k=1}^{N}$ as (column) words of length ℓ of some new 1-D constraint, which we define next.

The Nth Kronecker power of $G = (V, E, L)$, denoted by $G^{\otimes N} = (V^N, E^N, L^N)$, is defined as follows. The vertex set V^N is simply the Nth Cartesian power of V; that is

$$V^N = \{\langle v_1, v_2, \ldots, v_N\rangle : v_k \in V\}.$$

An edge $\boldsymbol{e} = \langle e_1, e_2, \ldots, e_N\rangle \in E^N$ goes from vertex $\boldsymbol{v} = \langle v_1, v_2, \ldots, v_N\rangle \in V^N$ to vertex $\boldsymbol{v}' = \langle v_1', v_2', \ldots, v_N'\rangle \in V^N$ and is labeled $L^N(\boldsymbol{e}) = \langle L(e_1), L(e_2), \ldots, L(e_N)\rangle$ whenever for all $1 \le k \le N$, e_k is an edge from v_k to v_k'.

Note that a path of length ℓ in $G^{\otimes N}$ is just a handy way to denote N paths of length ℓ in G. Accordingly, the $\ell \times N$ arrays $(\boldsymbol{\gamma}_k)_{k=1}^{N}$ are the words of length ℓ in $S(G^{\otimes N})$.

Let G be as in Section I and let $A_G = (a_{i,j})$ be the adjacency matrix of G. Denote by $\mathbf{1}$ the $1 \times |V|$ all-one row vector. The description of our M-track parallel encoder for $S = S(G)$ makes use of the following definition. A $|V| \times |V|$ nonnegative integer matrix $D = (d_{i,j})_{i,j \in V}$ is called a (valid) *multiplicity matrix* with respect to G and M if

$$\mathbf{1} \cdot D \cdot \mathbf{1}^T \le M \tag{3}$$

$$\mathbf{1} \cdot D = \mathbf{1} \cdot D^T \quad \text{and} \tag{4}$$

$$d_{i,j} > 0, \text{ only if } a_{i,j} > 0. \tag{5}$$

(While any multiplicity matrix will produce a parallel encoder, some will have higher rates than others. In Section IV, we show how to compute multiplicity matrices D that yield rates close to $\mathsf{cap}(S(G))$.)

Recall that we have at our disposal M tracks. However, we will effectively be using only the first $N = \mathbf{1} \cdot D \cdot \mathbf{1}^T$ tracks in order to encode information. The last $M - N$ tracks will all be equal to the first track, say.

Write $\boldsymbol{r} = (r_i)_{i \in V} = \mathbf{1} \cdot D^T$. A vertex $\boldsymbol{v} = \langle v_k\rangle_{k=1}^{N} \in V^N$ is a *typical vertex* (with respect to D) if for all i, the vertex $i \in V$ appears as an entry in $\boldsymbol{v}$ exactly r_i times. Also, an edge $\boldsymbol{e} = \langle e_k\rangle_{k=1}^{N} \in E^N$ is a *typical edge* with respect to D if for all $i, j \in V$, there are exactly $d_{i,j}$ entries e_k which—as edges in G—start at vertex i and terminate in vertex j.

A simple computation shows that the number of outgoing typical edges from a typical vertex equals

$$\Delta = \frac{\prod_{i \in V} r_i!}{\prod_{i,j \in V} d_{i,j}! \cdot a_{i,j}^{-d_{i,j}}} \tag{6}$$

(where $0^0 \triangleq 1$). For example, in the simpler case where G does not contain parallel edges ($a_{i,j} \in \{0, 1\}$), we are in effect counting in (6) permutations with repetitions, each time for a different vertex $i \in V$.

The encoding process will be carried out as follows. We start at some fixed typical vertex $\boldsymbol{v}^{(0)} \in V^N$. Out of the set of outgoing edges from $\boldsymbol{v}^{(0)}$, we consider only typical edges. The edge we choose to traverse is determined by the information bits. After traversing the chosen edge, we arrive at vertex $\boldsymbol{v}^{(1)}$. By (4), $\boldsymbol{v}^{(1)}$ is also a typical vertex, and the process starts over. This process defines an M-track parallel encoder for $S = S(G)$ at rate

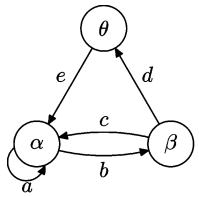
$$R = R(D) = \frac{\lfloor \log_2 \Delta \rfloor}{M}.$$

This encoder is $(\mathsf{m}, 0)$-SBD, where m is the memory of G.

Consider now how we map $M \cdot R$ information bits into an edge choice $\boldsymbol{e} \in E^N$ at any given stage t. Assuming again the simpler case of a graph with no parallel edges, a natural choice would be to use an instance of enumerative coding [8]. Specifically, suppose that for $0 \le \delta \le n$, a procedure for encoding information by an n-bit binary vector with Hamming weight δ were given. Suppose also that $V = \{1, 2, \ldots, |V|\}$. We could use this procedure as follows. First, for $n = r_1$ and $\delta = d_{1,1}$, the binary word given as output by the procedure will define which $d_{1,1}$ of the possible r_1 entries in $\boldsymbol{e}$ will be equal to the edge in E from the vertex $1 \in V$ to itself (if no such edge exists, then $d_{1,1} = 0$). Having chosen these entries, we run the procedure with $n = r_1 - d_{1,1}$ and $\delta = d_{1,2}$ to choose from the remaining $r_1 - d_{1,1}$ entries those that will contain the edge in E from $1 \in V$ to $2 \in V$. We continue this process, until all r_1 entries in $\boldsymbol{e}$ containing an edge outgoing from $1 \in V$ have been picked. Next, we run the procedure with $n = r_2$ and $\delta = d_{2,1}$, and so forth. The more general case of a graph containing parallel edges will include a preliminary step: encoding information in the choice of the $d_{i,j}$ edges used to traverse from i to j ($a_{i,j}$ options for each such edge).

A fast implementation of enumerative coding is presented in Section VII. The above-mentioned preliminary step makes use of the Schönhage–Strassen integer-multiplication algorithm [9, Sec. 7.5], and the resulting encoding time complexity is proportional[1] to $M \log^2 M \log\log M$. It turns out that this is also

[1]Actually, the time complexity for the preliminary step can be made linear in M, with a negligible penalty in terms of rate: Fix i and j, and let η be an integer design parameter. Assume for simplicity that $\eta | d_{i,j}$. The number of vectors of length η over an alphabet of size $a_{i,j}$ is obviously $a_{i,j}^{\eta}$. So, we can encode $\lfloor \eta \log_2 a_{i,j} \rfloor$ bits through the choice of such a vector. Repeating this process, we can encode $(d_{i,j}/\eta) \cdot \lfloor \eta \log_2 a_{i,j} \rfloor$ bits through the choice of $d_{i,j}/\eta$ such vectors. The concatenation of these vectors is taken to represent our choice of edges. Note that the encoding process is linear in M for constant η. Also, our losses (due to the floor function) become negligible for modestly large η.



$$A_G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Fig. 4. Running Example: Graph G and the corresponding adjacency matrix A_G.

the decoding time complexity. Further details are given in Section VII.

Section IV shows how to find a good multiplicity matrix, i.e., a matrix D such that $R(D)$ is close to $\mathsf{cap}(S(G))$.

IV. Computing a Good Multiplicity Matrix

In order to enhance the exposition of this section, we accompany it by a running example (see Fig. 4).

Throughout this section, we assume a probability distribution on the edges of G, which is the maxentropic stationary Markov chain $\mathcal{P}$ on G [1]. Without loss of generality, we can assume that G is irreducible (i.e., strongly connected), in which case $\mathcal{P}$ is indeed unique. Let the matrix $Q = (q_{i,j})$ be the transition matrix induced by $\mathcal{P}$, i.e., $q_{i,j}$ is the probability of traversing an edge from $i \in V$ to $j \in V$, conditioned on currently being at vertex $i \in V$.

Let $\boldsymbol{\pi} = (\pi_i)$ be the $1 \times |V|$ row vector corresponding to the stationary distribution on V induced by Q; namely, $\boldsymbol{\pi} Q = \boldsymbol{\pi}$ and $\sum_{i \in V} \pi_i = 1$. Let

$$M' = M - \lfloor |V| \operatorname{diam}(G)/2 \rfloor, \tag{7}$$

and define

$$\boldsymbol{\rho} = (\rho_i), \quad \rho_i = M' \pi_i, \text{ and } P = (p_{i,j}), \quad p_{i,j} = \rho_i q_{i,j}.$$

Example 1: Taking the number of tracks in our running example (Fig. 4) to be $M = 12$ gives $M' = 9$. Also, our running example has

$$\boldsymbol{\pi} = (\, 0.619 \quad 0.282 \quad 0.099 \,)$$

and

$$Q = \begin{pmatrix} 0.544 & 0.456 & 0 \\ 0.647 & 0 & 0.353 \\ 1 & 0 & 0 \end{pmatrix}.$$

Thus

$$\boldsymbol{\rho} = (\, 5.57 \quad 2.54 \quad 0.89 \,)$$

and

$$P = \begin{pmatrix} 3.03 & 2.54 & 0 \\ 1.65 & 0 & 0.89 \\ 0.89 & 0 & 0 \end{pmatrix}. \qquad \square$$

Note that

$$\boldsymbol{\rho} = \mathbf{1} \cdot P^T \quad \text{and} \quad M' = \mathbf{1} \cdot P \cdot \mathbf{1}^T.$$

Also, observe that (3)–(5) hold when we substitute P for D. Thus, if all entries of P were integers, then we could take D equal to P. In a way, that would be the best choice we could have made: by using Stirling's approximation,[2] we could deduce that $R(D)$ approaches $\mathsf{cap}(S(G))$ as $M \to \infty$. However, the entries of P, as well as $\boldsymbol{\rho}$, may be nonintegers.

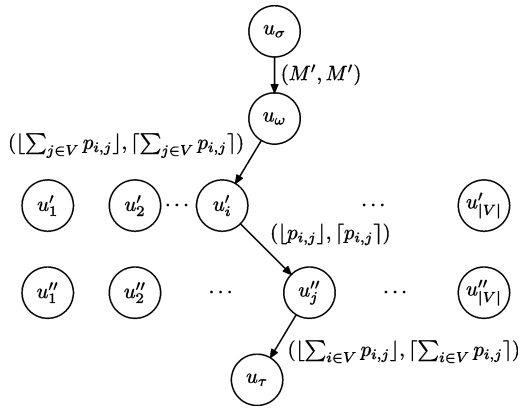


Fig. 5. Flow network for the proof of Lemma 2. An edge labeled (a, b) has lower and upper bounds a and b, respectively.

We say that an *integer* matrix $\tilde{P} = (\tilde{p}_{i,j})$ is a *good quantization* of $P = (p_{i,j})$ if

$$M' = \sum_{i,j \in V} p_{i,j} = \sum_{i,j \in V} \tilde{p}_{i,j} \tag{8}$$

$$\left\lfloor \sum_{j \in V} p_{i,j} \right\rfloor \le \sum_{j \in V} \tilde{p}_{i,j} \le \left\lceil \sum_{j \in V} p_{i,j} \right\rceil \tag{9}$$

$$\lfloor p_{i,j} \rfloor \le \tilde{p}_{i,j} \le \lceil p_{i,j} \rceil \tag{10}$$

and

$$\left\lfloor \sum_{i \in V} p_{i,j} \right\rfloor \le \sum_{i \in V} \tilde{p}_{i,j} \le \left\lceil \sum_{i \in V} p_{i,j} \right\rceil. \tag{11}$$

Namely, a given entry in $\tilde{P}$ is either the floor or the ceiling of the corresponding entry in P, and this also holds for the sum of entries of a given row or column in $\tilde{P}$; moreover, the sum of entries in both $\tilde{P}$ and P are exactly equal (to M').

Lemma 2: There exists a matrix $\tilde{P}$ which is a good quantization of P. Furthermore, such a matrix can be found by an efficient algorithm.

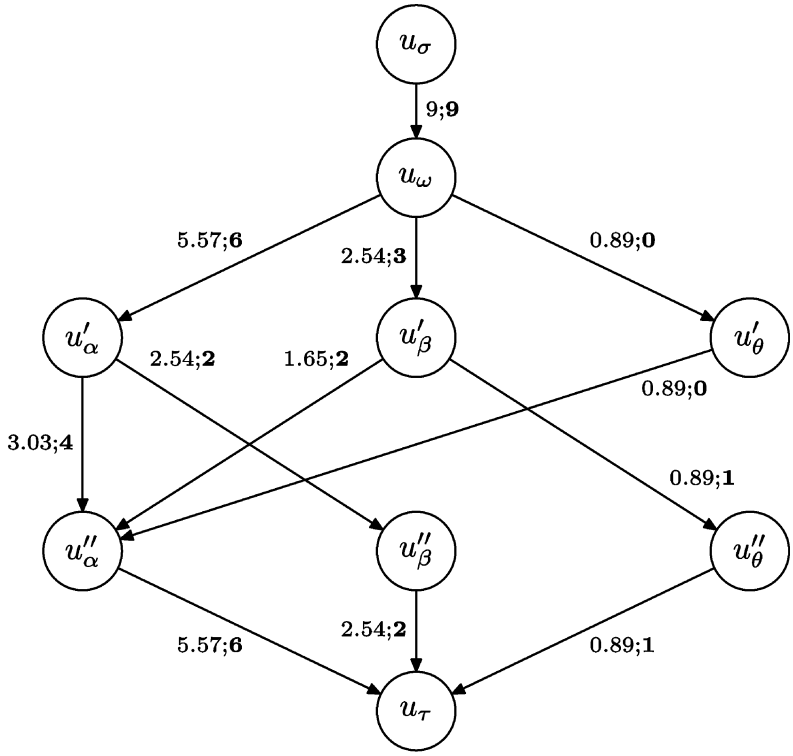
Proof: We recast (8)–(11) as an integer flow problem (see Figs. 5 and 6). Consider the following flow network, with upper and lower bounds on the flow through the edges [10, Sec. 6.7]. The network has the vertex set

$$\{u_\sigma\} \cup \{u_\omega\} \cup \{u_\tau\} \cup \{u'_i\}_{i \in V} \cup \{u''_j\}_{j \in V}$$

with source u_σ and target u_τ. Henceforth, when we refer to the upper (lower) bound of an edge, we mean the upper (lower) bound on the flow through it. There are four kinds of edges.

1) An edge $u_\sigma \to u_\omega$ with upper and lower bounds both equal to M'.
2) $u_\omega \to u'_i$ for every $i \in V$, with the upper and lower bounds $\left\lfloor \sum_{j \in V} p_{i,j} \right\rfloor$ and $\left\lceil \sum_{j \in V} p_{i,j} \right\rceil$, respectively.
3) $u'_i \to u''_j$ for every $i, j \in V$, with the upper and lower bounds $\lfloor p_{i,j} \rfloor$ and $\lceil p_{i,j} \rceil$, respectively.

[2] As will be done in the proof of Theorem 1, to prove a more general claim.



$$P = \begin{pmatrix} 3.03 & 2.54 & 0 \\ 1.65 & 0 & 0.89 \\ 0.89 & 0 & 0 \end{pmatrix}, \qquad \tilde{P} = \begin{pmatrix} 4 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Fig. 6. Running Example (continued): The flow network derived from P in Example 1. An edge labeled $a; \boldsymbol{b}$ has lower and upper bounds $\lfloor a \rfloor$ and $\lceil a \rceil$, respectively. A legal real flow is given by a. A legal integer flow is given by $\boldsymbol{b}$. The matrix $\tilde{P}$ resulting from the legal integer flow is given, as well as the matrix P (again).

4) $u''_j \to u_\tau$ for every $j \in V$, with the upper and lower bounds $\lfloor \sum_{i \in V} p_{i,j} \rfloor$ and $\lceil \sum_{i \in V} p_{i,j} \rceil$, respectively.

We claim that (8)–(11) can be satisfied if a legal integer flow exists: simply take $\tilde{p}_{i,j}$ as the flow on the edge from u'_i to u''_j.

It is well known that if a legal *real* flow exists for a flow network with integer upper and lower bounds on the edges, then a legal *integer* flow exists as well [10, Theorem 6.5]. Moreover, such a flow can be efficiently found [10, Sec. 6.7]. To finish the proof, we now exhibit such a legal real flow:

1) The flow on the edge $u_\sigma \to u_\omega$ is $\sum_{i,j \in V} p_{i,j} = M'$.
2) The flow on an edge $u_\omega \to u'_i$ is $\sum_{j \in V} p_{i,j}$.
3) The flow on an edge $u'_i \to u''_j$ is $p_{i,j}$.
4) The flow on an edge $u''_j \to u_\tau$ is $\sum_{i \in V} p_{i,j}$. □

For the remaining part of this section, we assume that $\tilde{P}$ is a good quantization of P (say, $\tilde{P}$ is computed by solving the integer flow problem in the last proof). The next lemma states that $\tilde{P}$ "almost" satisfies (4).

Lemma 3: Let $\tilde{\boldsymbol{\rho}} = (\tilde{\rho}_i) = \mathbf{1} \cdot \tilde{P}^T$ and $\tilde{\boldsymbol{r}} = (\tilde{r}_i) = \mathbf{1} \cdot \tilde{P}$. Then, for all $i \in V$

$$\tilde{\rho}_i - \tilde{r}_i \in \{-1, 0, 1\}.$$

Proof: From (9), we get that for all $i \in V$,

$$\left\lfloor \sum_{j \in V} p_{i,j} \right\rfloor \le \tilde{\rho}_i \le \left\lceil \sum_{j \in V} p_{i,j} \right\rceil. \tag{12}$$

Recall that (4) is satisfied if we replace D by P. Thus, by (11), we have that (12) also holds if we replace $\tilde{\rho}_i$ by $\tilde{r}_i$. We conclude that $|\tilde{\rho}_i - \tilde{r}_i| \le 1$. The proof follows from the fact that entries of $\tilde{P}$ are integers, and thus so are those of $\tilde{\boldsymbol{\rho}}$ and $\tilde{\boldsymbol{r}}$. □

The following lemma will be the basis for augmenting $\tilde{P}$ so that (4) is satisfied.

Lemma 4: Fix two distinct vertices $s, t \in V$. We can efficiently find a $|V| \times |V|$ matrix $F^{(s,t)} = F = (f_{i,j})_{i,j \in V}$ with nonnegative integer entries, such that the following three conditions hold.

(i)
$$\mathbf{1} \cdot F \cdot \mathbf{1}^T \le \operatorname{diam}(G).$$

(ii) For all $i, j \in V$
$$f_{i,j} > 0 \text{ only if } a_{i,j} > 0.$$

(iii) Denote $\boldsymbol{\xi} = \mathbf{1} \cdot F^T$ and $\boldsymbol{x} = \mathbf{1} \cdot F$. Then, for all $i \in V$
$$x_i - \xi_i = \begin{cases} -1, & \text{if } i = s \\ 1, & \text{if } i = t \\ 0, & \text{otherwise.} \end{cases}$$

Proof: Let $v_1 = s, v_2, v_3 \ldots, v_{\ell+1} = t$ be the vertices along a shortest path from s to t in G. For all $i, j \in V$, define

$$f_{i,j} = \left|\{1 \le h \le \ell : v_h = i \text{ and } v_{h+1} = j\}\right|. \tag{13}$$

Namely, $f_{i,j}$ is the number of edges from i to j along the path.

Conditions (i) and (ii) easily follow from (13). Condition (iii) follows from the fact that $\xi_i(x_i)$ is equal to the number of edges along the path for which i is the start (end) vertex of the edge. □

The matrix $\tilde{P}$ will be the basis for computing a good multiplicity matrix D, as we demonstrate in the proof of the next theorem.

Theorem 5: Let $\tilde{P} = (\tilde{p}_{i,j})$ be a good quantization of P. There exists a multiplicity matrix $D = (d_{i,j})$ with respect to G and M, such that

1) $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i, j \in V$, and
2) $M' \leq \mathbf{1} \cdot D \cdot \mathbf{1}^T \leq M$

(where M' is as defined in (7)). Moreover, the matrix D can be found by an efficient algorithm.

Proof: Consider a vertex $i \in V$. If $\tilde{r}_i > \tilde{\rho}_i$, then we say that vertex i has a *surplus* of $\tilde{r}_i - \tilde{\rho}_i$. In this case, by Lemma 3, we have that the surplus is equal to 1. On the other hand, if $\tilde{r}_i < \tilde{\rho}_i$ then vertex i has a *deficiency* of $\tilde{\rho}_i - \tilde{r}_i$, which again is equal to 1.

Of course, since $\sum_{i \in V} \tilde{\rho}_i = \sum_{i \in V} \tilde{r}_i = M'$, the total surplus is equal to the total deficiency, and both are denoted by Surp

$$\mathrm{Surp} = \sum_{i \in V} \max\{0, \tilde{r}_i - \tilde{\rho}_i\} = -\sum_{i \in V} \min\{0, \tilde{r}_i - \tilde{\rho}_i\}. \quad (14)$$

Denote the vertices with surplus as $(s_k)_{k=1}^{\mathrm{Surp}}$ and the vertices with deficiency as $(t_k)_{k=1}^{\mathrm{Surp}}$. Recalling the matrix F from Lemma 4, we define

$$D = \tilde{P} + \sum_{k=1}^{\mathrm{Surp}} F^{(s_k, t_k)}.$$

We first show that D is a valid multiplicity matrix. Note that $\mathrm{Surp} \leq |V|/2$. Thus, (3) follows from (7), (8), and (i). The definitions of surplus and deficiency vertices along with (iii) give (4). Finally, recall that (5) is satisfied if we replace $d_{i,j}$ by $p_{i,j}$. Thus, by (10), the same can be said for $\tilde{p}_{i,j}$. Combining this with (ii) yields (5).

Since the entries of $F^{(s_k, t_k)}$ are nonnegative for every k, we must have that $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i, j \in V$. This, together with (3) and (8), implies in turn that $M' \leq \mathbf{1} \cdot D \cdot \mathbf{1}^T \leq M$. □

Example 2: For the matrix $\tilde{P}$ of our running example in Fig. 6, we have

$$\tilde{\boldsymbol{r}} = (6 \quad 2 \quad 1), \quad \tilde{\boldsymbol{\rho}} = (6 \quad 3 \quad 0).$$

Thus, $\mathrm{Surp} = 1$. Namely, the vertex θ has a surplus while the vertex β has a deficiency. Taking $s = \theta$ and $t = \beta$ we get

$$F^{(s,t)} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 4 & 3 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \quad \square$$

Now that Theorem 5 is proved, we are in a position to prove our main result, Theorem 1. Essentially, the proof involves using the Stirling approximation and taking into account the various quantization errors introduced into D. The proof itself is given in the Appendix.

V. Enumerative Coding Into Sequences With a Given Markov Type

The main motivation for our methods is 2-D constrained coding. However, in this section, we show that they might be interesting in certain aspects of 1-D coding as well. Given a labeled graph G, a classic method for building an encoder for the 1-D constraint $S(G)$ is the state-splitting algorithm [11]. The rate of an encoder built by [11] approaches the capacity of $S(G)$. Also, the word the encoder outputs has a corresponding path in G, with the following favorable property: the probability of traversing a certain edge approaches the maxentropic probability of that edge (assuming an unbiased source distribution). However, what if we were to build an encoder with a different probability distribution on the edges? This scenario may occur, for example, when there is a requirement that all the output words of a given length N that are generated by the encoder have a prescribed Hamming weight.[3]

More formally, suppose that we are given a labeled graph $G = (V, E, L)$; to make the exposition simpler, suppose that G does not contain parallel edges. Let Q and $\boldsymbol{\pi}$ be a transition matrix and a stationary probability distribution corresponding to a stationary (but not necessarily maxentropic) Markov chain $\mathcal{P}$ on G. We assume without loss of generality (w.l.o.g.) that each edge in G has a positive conditional probability. We are also given an integer M, which we will shortly elaborate on.

We first describe our encoder in broad terms, so as that its merits will be obvious. Let D and N be as previously defined, and let $R_T(D)$ be specified shortly. We start at some fixed vertex $v_0 \in V$. Given $M \cdot R_T(D)$ information bits, we traverse a soon to be defined cyclic path of length N in G. The concatenation of the edge labels along the path is the word we output. Of course, since the path is cyclic, the concatenation of such words is indeed in $S(G)$. Moreover, the path will have the following key property: the number of times an edge from i to j is traversed equals $d_{i,j}$. Namely, if we uniformly pick one of the N edges of the path, the probability of picking a certain edge e is constant (not a function of the input bits), and is equal to the probability of traversing e on the Markov chain $\mathcal{P}$, up to a small quantization error. The rate R_T of our encoder will satisfy (1), where we replace R by R_T and $\mathsf{cap}(S)$ by the entropy of $\mathcal{P}$. We would like to be able to exactly specify the path length N as a design parameter. However, we specify M and get an N between M and $M - \lfloor |V| \operatorname{diam}(G)/2 \rfloor$.

Our encoding process will make use of an *oriented tree*, a term which we will now define. A set of edges $T \subseteq E$ is an oriented tree of G with root v_0 if $|T| = |V| - 1$ and for each $u \in V$ there exists a path from u to v_0 consisting entirely of edges in T (see Fig. 7). Note that if we reverse the edge directions of an oriented tree, we get a directed tree as defined in [13, Theorem 2.5]. Since reversing the directions of all edges in an irreducible graph results in an irreducible graph, we have by [13, Lemma 3.3] that an oriented tree T indeed exists in G, and can be efficiently found. So, let us fix some oriented tree T with

[3]We remark in passing that one may use convex programming techniques (see [12, Sec. V]) in order to efficiently solve the following optimization problem: find a probability distribution on the edges of G yielding a stationary Markov chain with largest possible entropy, subject to a set of edges (such as the set of edges with label "1") having a prescribed cumulative probability.

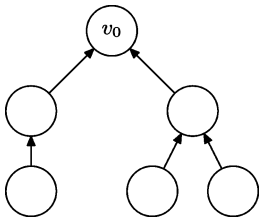


Fig. 7. Oriented tree with root v_0.

root v_0. By [13, Theorem 2.5], we have that every vertex $u \in V$ which is not the root v_0 has an out-degree equal to 1. Thus, for each such vertex u we may define $\mathrm{parent}(u)$ as the destination of the single edge in T going out of u.

We now elaborate on the encoding process. The encoding consists of two steps. In the first step, we map the information bits to a collection of lists. In the second step, we use the lists in order to define a cyclic path.

First step: Given $M \cdot R_T(D)$ information bits, we build for each vertex $i \in V$ a list $\boldsymbol{\lambda}^{(i)}$ of length r_i

$$\boldsymbol{\lambda}^{(i)} = \left(\lambda_1^{(i)}, \lambda_2^{(i)}, \ldots, \lambda_{r_i}^{(i)}\right).$$

The entries of each $\boldsymbol{\lambda}^{(i)}$ are vertices in V. Moreover, the following properties are satisfied for all i.

- The number of times j is an entry in $\boldsymbol{\lambda}^{(i)}$ is exactly $d_{i,j}$.
- If $i \neq v_0$, then the last entry of the list equals the parent of i. Namely

$$\lambda_{r_i}^{(i)} = \mathrm{parent}(i).$$

Recalling (6), a simple calculation shows that the number of possible list collections is

$$\Delta_T = \Delta \cdot \prod_{i \in V \setminus \{v_0\}} \frac{d_{i,\mathrm{parent}(i)}}{r_i}. \tag{15}$$

Thus, we define the rate of encoding as

$$R_T = \frac{\lfloor \log_2 \Delta_T \rfloor}{M}.$$

Also, note that as in the 2-D case, we may use enumerative coding in order to efficiently map information bits to lists.

Second step: We now use the lists $\boldsymbol{\lambda}^{(i)}$, $i \in V$, in order to construct a cyclic path starting at vertex v_0. We start the path at v_0 and build a length-N path according to the following rule: when exiting vertex i for the kth time, traverse the edge going into vertex $\lambda_k^{(i)}$.

Of course, our encoding method is valid (and invertible) iff we may always abide by the above-mentioned rule. Namely, we do not get "stuck," and manage to complete a cyclic path of length N. This is indeed the case: define an auxiliary graph $G(D)$ with the same vertex set, V, as G and $d_{i,j}$ parallel edges from i to j (for all $i, j \in V$). First, recall that for sufficiently large M, the presence of an edge from i to j in G implies that $d_{i,j} > 0$. Thus, since G was assumed to be irreducible, $G(D)$ is irreducible as well. Also, an edge in T from i to j implies the existence of an edge in $G(D)$ from i to j. Second, note that by (4), the number of times we are supposed to exit a vertex is equal to the number of times we are supposed to enter it. The rest of the proof follows from [14, p. 56, Claim 2], applied to the auxiliary graph $G(D)$. Namely, our encoder follows directly from van Aardenne-Ehrenfest and de Bruijn's [15] theorem on counting Eulerian cycles in a graph.

We now return to the rate, R_T, of our encoder. From (7), (10), (11), and Theorem 5, we see that for M sufficiently large, Δ_T is greater than some positive constant times Δ. Thus, (1) still holds if we replace R by R_T and $\mathsf{cap}(S)$ by the entropy of $\mathcal{P}$.

VI. An Example, and Two Improvement Techniques

Recall from Section II the kings constraint: its elements are all the binary arrays in which no two "1" symbols are adjacent on a row, column, or diagonal. By employing the methods presented in [16], we may calculate an upper bound on the rate of the constraint. This turns out to be 0.425078. We will show an encoding/decoding method with rate slightly larger than 0.396 (about 93% of the upper bound). In order to do this, we assume that the array has 100,000 columns. Our encoding method has a fixed rate and has a vertical window of size 2 and vertical anticipation 0.

We should point out now that a straightforward implementation of the methods we have previously defined gives a rate which is strictly *less* than 0.396. Namely, this section also outlines two improvement techniques which help boost the rate.

We start out as in the example given in Section II, except that the width of the data strips is now $w = 9$ (the width of the merging strips remains $b = 1$). The graph G we choose produces all width-w arrays satisfying the kings constraint, and we take the merging strips to be all-zero. Our array has 100,000 columns, so we have $M = 10{,}000$ tracks (the last, say, column of the array will essentially be unused; we can set all of its values to 0).

Define the normalized capacity as

$$\frac{\mathsf{cap}(S(G))}{w+b}.$$

The graph G has $|V| = 89$ vertices and normalized capacity

$$\frac{\mathsf{cap}(S(G))}{w+b} \approx \frac{\mathsf{cap}(S(G))}{w+b(1-1/M)} \approx 0.402.$$

This number is about 94.5% from the upper bound on the capacity of our 2-D constraint. Thus, as expected, there is an inherent loss in choosing to model the 2-D constraint as an essentially 1-D constraint. Of course, this loss can be made smaller by increasing w (but the graph G will grow as well).

From Theorem 1, the rate of our encoder will approach the normalized capacity of 0.402 as the number of tracks M grows. So, once the graph G is chosen, the parameter we should be comparing it to is the normalized capacity. We now apply the methods defined in Section IV and find a multiplicity matrix D. Recall that the matrix D defines an encoder. In our case, this encoder has a rate of about 0.381. This is 94% of the normalized capacity, and is quite disappointing (but the improvements shown in Sections VI-A and B below are going to improve this rate). On the other hand, note that if we had limited ourselves to encode to each track independently of the others, then the best

rate we could have hoped for with 0 vertical anticipation would turn out to be 0.3 (see [17, Theorem 5]).

A. Moore-Style Reduction

We now define a graph G which we call the reduction of G. Essentially, we will encode by constructing paths in G, and then translate these to paths in G. In both G and G, the maxentropic distributions have the same entropy. The main virtue of G is that it often has fewer vertices and edges compared to G. Thus, the penalty in (1) resulting from using a finite number of tracks will often be smaller.

For $s \geq 0$, we now recursively define the concept of s-equivalence (very much like in the Moore algorithm [1, p. 1660]).

- For $s = 0$, any two vertices $v_1, v_2 \in V$ are 0-equivalent.
- For $s > 0$, two vertices $v_1, v_2 \in V$ are s-equivalent iff 1) the two vertices v_1, v_2 are $(s-1)$-equivalent, and 2) for each $(s-1)$-equivalence class c, the number of edges from v_1 to vertices in c is equal to the number of edges from v_2 to vertices in c.

Denote by Π_s the partition induced by s-equivalence. For the graph G given in Fig. 3

$$\Pi_0 = \{0000,0001,0010,0100,0101,1000,1001,1010\}$$
$$\Pi_{s\geq 1} = \{0000\},\{0010,0100\},\{1000,0001\},\{1010,1001,0101\}.$$

Note that, by definition, Π_{s+1} is a refinement of Π_s. Thus, let s' be the smallest s for which $\Pi_s = \Pi_{s+1}$. The set $\Pi_{s'}$ can be efficiently found (essentially, by the Moore algorithm [1, p. 1660]).

Define a (nonlabeled) graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ as follows. The vertex set of G is

$$\mathsf{V} = \Pi_{s'}.$$

For each $\mathsf{c} \in \mathsf{V}$, let $v(\mathsf{c})$ be a fixed element of c (if c contains more than one vertex, then pick one arbitrarily). Also, for each $v \in V$, let $\mathsf{c}(v)$ be the class $\mathsf{c} \in \mathsf{V}$ such that $v \in \mathsf{c}$. Let $\sigma_G(e)$ ($\sigma_{\mathsf{G}}(e)$) and $\tau_G(e)$ ($\tau_{\mathsf{G}}(e)$) denote the start and end vertex of an edge e in G (G), respectively. The edge set E is defined as

$$\mathsf{E} = \bigcup_{\mathsf{c}\in\mathsf{V}} \{e \in E : \sigma_G(e) = v(\mathsf{c})\} \tag{16}$$

where

$$\sigma_{\mathsf{G}}(e) = \mathsf{c}(\sigma_G(e)) \quad \text{and} \quad \tau_{\mathsf{G}}(e) = \mathsf{c}(\tau_G(e)).$$

Namely, the number of edges from c_1 to c_2 in G is equal to the number of edges in G from some fixed $v_1 \in \mathsf{c}_1$ to elements of c_2, and, by the definition of s', this number does not depend on the choice of v_1. The graph G is termed the *reduction* of G. The reduction of G from Fig. 3 is given in Fig. 8. Note that since G was assumed to be irreducible, we must have that G is irreducible as well.

Lemma 6: The entropies of the maxentropic Markov chains on G and G are equal.

Proof: Let $\mathsf{A} = \mathsf{A}_{\mathsf{G}}$ be the adjacency matrix of G, and recall that $A = A_G$ is the adjacency matrix of G. Let λ' and $\boldsymbol{x}' = (x'_{\mathsf{c}})_{\mathsf{c}\in\mathsf{V}}$ be the Perron eigenvalue and right Perron eigenvector of A, respectively [1, Sec. 3.1]. Next, define the vector $\boldsymbol{x} = (x_v)_{v\in V}$ as

$$x_v = x'_{\mathsf{c}(v)}.$$

It is easily verifiable that $\boldsymbol{x}$ is a right eigenvector of A, with eigenvalue λ'. Now, since $\boldsymbol{x}'$ is a Perron eigenvector of an irreducible matrix, each entry of it is positive. Thus, each entry of $\boldsymbol{x}$ is positive as well. Since A is irreducible, we must have that $\boldsymbol{x}$ is a Perron eigenvector of A. So, the Perron eigenvalue of A is also λ'. □

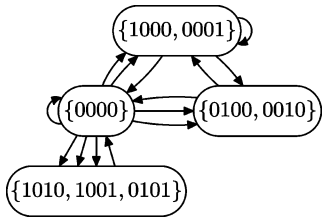


Fig. 8. Reduction of the graph G from Fig. 3.

The next lemma essentially states that we can think of paths in G as if they were paths in G.

Lemma 7: Let $\ell \geq 1$. Fix some $\mathsf{c}_0, \mathsf{c}_{\ell+1} \in \mathsf{V}$, and $v_0 \in \mathsf{c}_0$. There exists a one-to-one correspondence between the following sets. First set: paths of length ℓ in G with start vertex c_0 and end vertex $\mathsf{c}_{\ell+1}$. Second set: paths of length ℓ in G with start vertex v_0 and end vertex in $\mathsf{c}_{\ell+1}$.

Moreover, for $1 \leq t \leq \ell - 1$, the first t edges in a path belonging to the second set are a function of only the first t edges in the respective path in the first set.

Proof: We prove this by induction on ℓ. For $\ell = 1$, we have

$$\left|\{e \in \mathsf{E} : \sigma_{\mathsf{G}}(e) = \mathsf{c}_0, \quad \tau_{\mathsf{G}}(e) = \mathsf{c}_1\}\right| = \left|\{e \in E : \sigma_G(e) = v_0, \quad \tau_G(e) \in \mathsf{c}_1\}\right|.$$

To see this, note that we can assume w.l.o.g. that $v_0 = v(\mathsf{c}_0)$, and then recall (16). For $\ell > 1$, combine the claim for $\ell - 1$ with that for $\ell = 1$. □

Notice that $\operatorname{diam}(\mathsf{G}) \leq \operatorname{diam}(G)$. We now show why G is useful.

Theorem 8: Let D be the multiplicity matrix found by the methods previously outlined, where we replace G by G. Let $\mathsf{N} = \mathbf{1} \cdot \mathsf{D} \cdot \mathbf{1}^T$. We may efficiently encode (and decode) information to $G^{\otimes \mathsf{N}}$ in a row-by-row manner at rate $R(\mathsf{D})$.

Proof: We conceptually break our encoding scheme into two steps. In the first step, we "encode" (map) the information into N paths in G, each path having length ℓ. We do this as previously outlined (through typical vertices and edges in G). Note that this step is done at a rate of $R(\mathsf{D})$. In the second step, we map each such path in G to a corresponding path in G. By Lemma 7, we can indeed do this (take c_0 as the first vertex in the path, $\mathsf{c}_{\ell+1}$ as the last vertex, and $v_0 = v(\mathsf{c}_0)$).

By Lemma 7 we see that this two-step encoding scheme can easily be modified into one that is row-by-row. □

Applying the reduction to our running example (kings constraint with $w = 9$ and $b = 1$), reduces the number of vertices

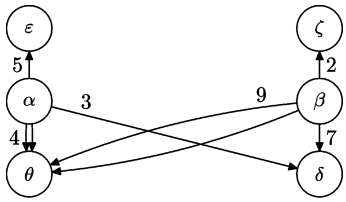


Fig. 9. Break–merge example graph.

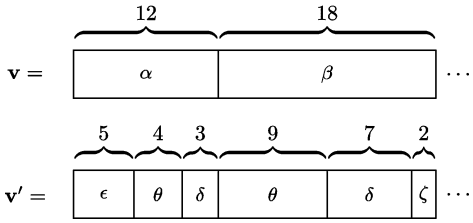


Fig. 10. Illustration of the entries in two typical vertices $\boldsymbol{v}$, $\boldsymbol{v}'$, where we got from $\boldsymbol{v}$ to $\boldsymbol{v}'$ by the standard encoding process.

from 89 in G to 34 in G. The computed D increases the rate to about 0.392, which is 97.5% of the normalized capacity.

B. Break–Merge

Let $\mathsf{G}^{\otimes \mathsf{N}}$ be the Nth Kronecker power of the Moore-style reduction G. Recall that the rate of our encoder is

$$R(\mathsf{D}) = \frac{\lfloor \log_2 \Delta \rfloor}{M}$$

where Δ is the number of typical edges in $\mathsf{G}^{\otimes \mathsf{N}}$ going out of a typical vertex. The second improvement involves expanding the definition of a typical edge, thus increasing Δ. This is best explained through an example. Suppose that Fig. 9 is a subgraph in G; namely, we show all edges going out of vertices α and β. Also, let the numbers next to the edges be equal to the corresponding entries in D. The main thing to notice at this point is that if the edges to ϵ and ζ are deleted ("break"), then α and β have exactly the same number of edges from them to vertex j, for all $j \in V$ (after the deletion of edges, vertices α and β can be "merged").

Let $\boldsymbol{v}$ be a typical vertex. A short calculation shows that the number of entries in $\boldsymbol{v}$ that are equal to α (β) is $5+4+3 = 12$ ($9+7+2 = 18$). Recall that the standard encoding process consists of choosing a typical edge $\boldsymbol{e}$ going out of the typical vertex $\boldsymbol{v}$ and into another typical vertex $\boldsymbol{v}'$. We now briefly review this process. Consider the 12 entries in $\boldsymbol{v}$ that are equal to α. The encoding process with respect to them will be as follows (see Fig. 10).

- Out of these 12 entries, choose five for which the corresponding entry in $\boldsymbol{v}'$ will be ε. Since there is exactly one edge from α the ε in G, the corresponding entries in $\boldsymbol{e}$ must be equal to that edge.
- Next, from the remaining seven entries, choose four for which the corresponding entries in $\boldsymbol{v}'$ will be θ. There are two parallel edges from α to θ, so choose which one to use in the corresponding entries in $\boldsymbol{e}$.
- We are left with three entries, the corresponding entries in $\boldsymbol{v}'$ will be δ. Also, we have one option as to the corresponding entries in $\boldsymbol{e}$.

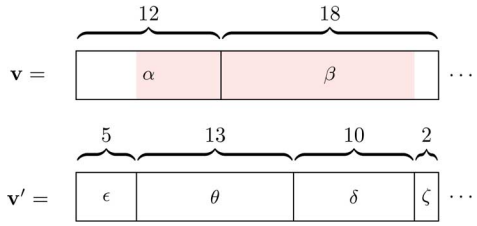


Fig. 11. Illustration of the entries in two typical vertices $\boldsymbol{v}$, $\boldsymbol{v}'$, where we got from $\boldsymbol{v}$ to $\boldsymbol{v}'$ by the improved encoding process. The shaded part corresponds to vertices that were merged.

A similar process is applied to the entries in $\boldsymbol{v}$ that are equal to β. Thus, the total number of options with respect to these entries is

$$\frac{12! \cdot 2^4}{5! \cdot 4! \cdot 3!} \cdot \frac{18! \cdot 2^9}{2! \cdot 9! \cdot 7!} \approx 3.97 \cdot 10^{14}.$$

Next, consider a different encoding process (see Fig. 11).

- Out of the 12 entries in $\boldsymbol{v}$ that are equal to α, choose five for which the corresponding entry in $\boldsymbol{v}'$ will be ε. As before, the corresponding entries in $\boldsymbol{e}$ have only one option.
- Out of the 18 entries in $\boldsymbol{v}$ that are equal to β, choose two for the corresponding entry in $\boldsymbol{v}'$ will be ζ. Again, one option for entries in $\boldsymbol{e}$.
- Now, of the remaining 23 entries in $\boldsymbol{v}$ that are equal to α or β, choose $4+9 = 13$ for which the corresponding entry in $\boldsymbol{v}'$ will be θ. We have two options for the entries in $\boldsymbol{e}$.
- We are left with $3+7 = 10$ entries in $\boldsymbol{v}$ that are equal to α or β. These will have δ as the corresponding entry in $\boldsymbol{v}'$, and one option in $\boldsymbol{e}$.

Thus, the total number of options is now

$$\binom{12}{5} \cdot \binom{18}{2} \cdot \frac{23! \cdot 2^{13}}{13! \cdot 10!} \approx 1.14 \cdot 10^{15}.$$

The important thing to notice is that in both cases, we arrive at a typical vertex $\boldsymbol{v}'$.

To recap, we first "broke" the entries in $\boldsymbol{v}$ that are equal to α into two groups: Those which will have ε as the corresponding entry in $\boldsymbol{v}'$ and those which will have θ or δ as the corresponding entry. Similarly, we broke entries in $\boldsymbol{v}$ that are equal to β into two groups. Next, we noticed that of these four groups, two could be "merged," since they were essentially the same. Namely, removing some edges from the corresponding vertices in G resulted in vertices which were mergeable.

Of course, these operations can be repeated. The hidden assumption is that the sequence of breaking and merging is fixed, and known to both the encoder and decoder. The optimal sequence of breaking and merging is not known to us. We used a heuristic. Namely, choose two vertices such that the sets of edges emanating from both have a large overlap. Then, break and merge accordingly. This was done until no breaking or merging was possible. We got a rate of about 0.396, which is 98.5% of the normalized capacity.

VII. Fast Enumerative Coding

Recall from Section III that in the course of our encoding algorithm, we make use of a procedure which encodes information into fixed-length binary words of constant weight. A way

to do this would be to use enumerative coding [8]. Immink [18] showed a method to significantly improve the running time of an instance of enumerative coding, with a typically negligible penalty in terms of rate. We now briefly show how to tailor Immink's method to our needs.

Denote by n and δ the length and Hamming weight, respectively, of the binary word we encode into. Some of our variables will be *floating-point* numbers with a mantissa of μ bits and an exponent of ϵ bits: each floating-point number is of the form $\overline{x} = s \cdot 2^t$ where s and t are integers such that

$$2^\mu \le s < 2^{\mu+1} \quad \text{and} \quad -2^{\epsilon-1} \le t < 2^{\epsilon-1}.$$

Note that $\mu + \epsilon$ bits are needed to store such a number. Also, note that every positive real x such that

$$2^\mu \cdot 2^{-2^{\epsilon-1}} \le x \le (2^{\mu+1} - 1) \cdot 2^{2^{\epsilon-1}-1}$$

has a floating-point approximation $\overline{x}$ with relative precision

$$\left(1 - \frac{1}{2^\mu}\right) \le \frac{\overline{x}}{x} \le \left(1 + \frac{1}{2^\mu}\right). \tag{17}$$

We assume the presence of two lookup tables. The first will contain the floating-point approximations of $1!, 2!, \ldots, n!$. The second will contain the floating-point approximations of $f(0), f(1), \ldots, f(\delta)$, where

$$f(\chi) = f_\mu(\chi) = 1 - \frac{32\chi + 16}{2^\mu}.$$

In order to exclude uninteresting cases, assume that $\mu \ge 10$ and is such that $f(\delta) \ge 1/2$. Also, take ϵ large enough so that $n!$ is less than the maximum number we can represent by floating point. Thus, we can assume that $\mu = O(\log \delta)$ and $\epsilon = O(\log n)$.

Notice that in our case, we can bound both n and δ from above by the number of tracks M. Thus, we will actually build beforehand two lookup tables of size $2M(\mu + \epsilon)$ bits.

Let $\overline{x}$ denote the floating-point approximation of x, and let $*$ and $\div$ denote floating-point multiplication and division, respectively. For $0 \le \chi \le \kappa \le n$, we define

$$\left\lceil \frac{\kappa}{\chi} \right\rceil = \left\lceil (\overline{\kappa!} * \overline{f(\chi)}) \div (\overline{\chi!} * \overline{(\kappa - \chi)!}) \right\rceil.$$

Note that since we have stored the relevant numbers in our lookup table, the time needed to calculate the above function is only $O(\mu^2 + \epsilon)$. The encoding procedure is given in Fig. 12. We note the following points.

- The variables n, ψ, δ, and ι are integers (as opposed to floating-point numbers).
- In the subtraction of $\lceil \frac{n-\iota}{\delta-1} \rceil$ from ψ in line 5, the floating-point number $\lceil \frac{n-\iota}{\delta-1} \rceil$ is "promoted" to an integer (the result is an integer).

We must now show that the procedure is valid, namely, that given a valid input, we produce a valid output. For our procedure, this reduces to showing two things: 1) If the stopping condition is not met, a recursive call will be made. 2) The recursive call is given valid parameters as well. Namely, in the recursive call, ψ is nonnegative. Also, for the encoding to be invertible, we must further require that 3) $\lceil \frac{n}{0} \rceil = 1$ for $n \ge 0$.

Name: EnumEncode(n, δ, ψ)

Input: Integers n, δ, ψ such that $0 \le \delta \le n$ and $0 \le \psi < \lceil \frac{n}{\delta} \rceil$.
Output: A binary word of length n and weight δ.

```
if (δ == 0) // stopping condition:                          /* 1 */
   return 00...0 (n);                                      /* 2 */
for (ι ← 1; ι ≤ n − δ + 1; ι++) {                          /* 3 */
   if (ψ ≥ ⌈(n−ι)/(δ−1)⌉)                                  /* 4 */
      ψ ← ψ − ⌈(n−ι)/(δ−1)⌉;                               /* 5 */
   else                                                    /* 6 */
      return 00..01 (ι−1) ‖ EnumEncode(n − ι, δ − 1, ψ);   /* 7 */
}                                                          /* 8 */
```

Fig. 12. Enumerative encoding procedure for constant-weight binary words.

Condition 2 is clearly met, because of the check in line 4. Denote

$$\left\langle \frac{\kappa}{\chi} \right\rangle = (\overline{\kappa!} * \overline{f(\chi)}) \div (\overline{\chi!} * \overline{(\kappa - \chi)!})$$

(and so, $\lceil \frac{\kappa}{\chi} \rceil = \left\lceil \langle \frac{\kappa}{\chi} \rangle \right\rceil$). Condition 3) follows from the next lemma.

Lemma 9: Fix $0 \le \delta \le n$. Then

$$\binom{n}{\delta} \cdot \left(1 - \frac{32(\delta+1)}{2^\mu}\right) \le \left\langle \frac{n}{\delta} \right\rangle \le \binom{n}{\delta} \cdot \left(1 - \frac{32\delta}{2^\mu}\right).$$

Proof: The proof is essentially repeated invocations of (17) at the various stages of computation. We leave the details to the reader. □

Finally, Condition 1 follows easily from the next lemma.

Lemma 10: Fix $0 \le \delta \le n$. Then

$$\left\lceil \frac{n}{\delta} \right\rceil \le \sum_{\iota=1}^{n-\delta+1} \left\lceil \frac{n-\iota}{\delta-1} \right\rceil.$$

Proof: The claim will follow if we show that

$$\left\langle \frac{n}{\delta} \right\rangle \le \sum_{\iota=1}^{n-\delta+1} \left\langle \frac{n-\iota}{\delta-1} \right\rangle.$$

This is immediate from Lemma 9 and the binomial identity

$$\binom{n}{\delta} = \sum_{\iota=1}^{n-\delta+1} \binom{n-\iota}{\delta-1}$$ □.

Note that the penalty in terms of rate one suffers because of using our procedure (instead of plain enumerative coding) is negligible. Namely, $\log_2 \lceil \frac{n}{\delta} \rceil$ can be made arbitrarily close to $\log_2 \binom{n}{\delta}$. Since we take $\epsilon = O(\log n)$ and $\mu = O(\log \delta)$, we can show by amortized analysis that the running time of the procedure is $O(n \log^2 n)$. Specifically, see [19, Sec. 17.3], and take the potential of the binary vector corresponding to ψ as the number of entries in it that are equal to "0." The decoding procedure is a straightforward "reversal" of the encoding procedure, and its running time is also $O(n \log^2 n)$.

APPENDIX
PROOF OF THEOREM 1

Let $\tilde{\Delta}$ be as in (6), where we replace $d_{i,j}$ by $\tilde{p}_{i,j}$ and r_i by $\tilde{\rho}_i$. By the combinatorial interpretation of (6), and the fact that $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i, j \in V$, it easily follows that $\Delta \geq \tilde{\Delta}$. Thus

$$R(D) \geq \frac{\lfloor \log_2 \tilde{\Delta} \rfloor}{M} = \frac{M'}{M} \cdot \frac{\lfloor \log_2 \tilde{\Delta} \rfloor}{M'}.$$

Denote by e the base of natural logarithms. By Stirling's formula we have

$$\log_2(t!) = t \log_2(t/\mathsf{e}) + O(\log t)$$

and from (6) we get that

$$\log_2 \tilde{\Delta} = \sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i/\mathsf{e}) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}/\mathsf{e}) + \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(a_{i,j}) - O(|V|^2 \log M).$$

By (8) and (10)

$$\sum_{i,j \in V} \tilde{p}_{i,j} \log_2(a_{i,j}) = \sum_{i,j \in V} p_{i,j} \log_2(a_{i,j}) - O\left(|V|^2 \log(a_{\max}/a_{\min})\right).$$

Since $\sum_j \tilde{p}_{i,j} = \tilde{\rho}_i$, we have

$$\sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i/\mathsf{e}) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}/\mathsf{e}) = \sum_{i \in V} \tilde{\rho}_i \log_2(\tilde{\rho}_i) - \sum_{i,j \in V} \tilde{p}_{i,j} \log_2(\tilde{p}_{i,j}).$$

Moreover, by (9) and (10), the right-hand side of the last equation equals

$$\sum_{i \in V} \rho_i \log_2(\rho_i) - \sum_{i,j \in V} p_{i,j} \log_2(p_{i,j}) - O(|V|^2 \log M).$$

We conclude that

$$\log_2 \tilde{\Delta} = \sum_{i \in V} \rho_i \log_2(\rho_i) - \sum_{i,j \in V} p_{i,j} \log_2(p_{i,j}) + \sum_{i,j \in V} p_{i,j} \log_2(a_{i,j}) - O\left(|V|^2 (\log M \cdot a_{\max}/a_{\min})\right).$$

Finally, recall that $\rho_i = M' \pi_i$ and $p_{i,j} = \rho_i q_{i,j}$. Thus

$$\log_2 \tilde{\Delta} = M' H(\mathcal{P}) - O\left(|V|^2 (\log M \cdot a_{\max}/a_{\min})\right)$$

where $H(\mathcal{P})$ is the entropy of the stationary Markov chain $\mathcal{P}$ with transition matrix Q. Recall that $\mathcal{P}$ was selected to be max-entropic: $H(\mathcal{P}) = \mathsf{cap}(S(G))$. This fact, along with (7) and a short calculation, finishes the proof. □

ACKNOWLEDGMENT

The first author would like to thank Roee Engelberg for very helpful discussions.

REFERENCES

[1] B. H. Marcus, R. M. Roth, and P. H. Siegel, "Constrained systems and coding for recording channels," in *Handbook of Coding Theory*, V. Pless and W. Huffman, Eds. Amsterdam, The Netherlands: Elsevier, 1998, pp. 1635–1764.

[2] T. Etzion, "Cascading methods for runlength-limited arrays," *IEEE Trans. Inf. Theory*, vol. 43, no. 1, pp. 319–324, Jan. 1997.

[3] S. Halevy and R. M. Roth, "Parallel constrained coding with application to two-dimensional constraints," *IEEE Trans. Inf. Theory*, vol. 48, no. 5, pp. 1009–1020, May 2002.

[4] A. Kato and K. Zeger, "On the capacity of two-dimensional run-length constrained code," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1527–1540, Jul. 1999.

[5] W. Weeks and R. E. Blahut, "The capacity and coding gain of certain checkerboard codes," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1193–1203, May 1998.

[6] R. Burton and J. E. Steif, "Non-uniqueness of measures of maximal entropy for subshifts of finite type," *Ergod. Theory Dynam. Syst.*, vol. 14, pp. 213–235, 1994.

[7] J. C. de Souza, B. H. Marcus, R. New, and B. A. Wilson, "Constrained systems with unconstrained positions," *IEEE Trans. Inf. Theory*, vol. 48, no. 4, pp. 866–879, Apr, 2002.

[8] T. M. Cover, "Enumerative source coding," *IEEE Trans. Inf. Theory*, vol. IT-19, no. 1, pp. 73–77, Jan. 1973.

[9] A. V. Aho, J. E. Hopcroft, and J. de Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[10] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*. Engelwood Cliffs, NJ: Prentice-Hall, 1993.

[11] R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes—An application of symbolic dynamics to information theory," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 1, pp. 5–22, Jan. 1983.

[12] B. H. Marcus and R. M. Roth, "Improved Gilbert-Varshamov bound for constrained systems," *IEEE Trans. Inf. Theory*, vol. 38, no. 4, pp. 1213–1221, Jul. 1992.

[13] S. Even, *Graph Algorithms*. Cupertino, CA: Computer Sci. Press, 1979.

[14] R. P. Stanley, *Enumerative Combinatorics*. Cambridge, U.K.: Cambridge Univ. Press, 1999, vol. 2.

[15] T. v. Aardenne-Ehrenfest and N. G. de Bruijn, "Circuits and trees in oriented linear graphs," *Simon Stevin*, vol. 28, pp. 203–217, 1951.

[16] N. Calkin and H. S. Wilf, "The number of independent sets in a grid graph," *SIAM J. Discr. Math.*, vol. 11, pp. 54–60, 1997.

[17] B. H. Marcus and R. M. Roth, "Bounds on the number of states in encoder graphs for input-constrained channels," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 742–758, May 1991.

[18] K. A. S. Immink, "A practical method for approaching the channel capacity of constrained channels," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1389–1399, Sep. 1997.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

Ido Tal (S'05–M'08) was born in Haifa, Israel, in 1975. He received the B.Sc., M.Sc., and Ph.D. degrees in computer science from Technion–Israel Institute of Technology, Haifa, in 1998, 2003, and 2009, respectively.

He is a Postdoctoral Scholar at the Center for Magnetic Recording Research (CMRR), University of California, San Diego, La Jolla, CA. His research interests include constrained coding and error-control coding.

Tuvi Etzion (M'89–SM'99–F'04) was born in Tel-Aviv, Israel, in 1956. He received the B.A., M.Sc., and D.Sc. degrees from the Technion–Israel Institute of Technology, Haifa, Israel, in 1980, 1982, and 1984, respectively.

Since 1984 he has held a position in the Department of Computer Science at the Technion, where he is now a Professor. During the years 1986–1987, he was Visiting Research Professor with the Department of Electrical Engineering–Systems at the University of Southern California, Los Angeles. During the summers of 1990 and 1991 he was visiting Bellcore in Morristown, NJ. During the years 1994–1996, he was a Visiting Research Fellow in the Computer Science Department at Royal Holloway College, Egham, U.K. He also had several visits to the Coordinated Science Laboratory at University of Illinois in Urbana-Champaign during the years 1995–1998, two visits to HP Bristol, Bristol, U.K. during the summers of 1996, 2000, a few visits to the Department of Electrical Engineering, University of California, San Diego during the years 2000–2009, and several visits to the Mathematics Department at Royal Holloway College, Egham, U.K. during the years 2007–2008. His research interests include applications of discrete mathematics to problems in computer science and information theory, coding theory, and combinatorial designs.

Dr. Etzion is currently an Associate Editor for Coding Theory of the IEEE TRANSACTIONS ON INFORMATION THEORY.

Ron M. Roth (M'88–SM'97–F'03) was born in Ramat-Gan, Israel, in 1958. He received the B.Sc. degree in computer engineering, the M.Sc. degree in electrical engineering, and the D.Sc. degree in computer science from Technion–Israel Institute of Technology, Haifa, Israel, in 1980, 1984, and 1988, respectively.

Since 1988, he has been with the Computer Science Department at Technion, where he now holds the General Yaakov Dori Chair in Engineering. During the academic years 1989–1991 he was a Visiting Scientist at IBM Research Division, Almaden Research Center, San Jose, CA, and during 1996–1997 and 2004–2005, he was on sabbatical leave at Hewlett-Packard Laboratories, Palo Alto, CA. He is the author of the book *Introduction to Coding Theory*, published by Cambridge University Press in 2006. His research interests include coding theory, information theory, and their application to the theory of complexity.

Dr. Roth was an Associate Editor for Coding Theory for IEEE TRANSACTIONS ON INFORMATION THEORY from 1998 until 2001, and he is now serving as an Associate Editor for *SIAM Journal on Discrete Mathematics*.