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

Ido Tal Tuvi Etzion Ron M. Roth
Computer Science Department,
Technion, Haifa 32000, Israel.
Email: {idotal, etzion, ronny}@cs.technion.ac.il

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.

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 [8, §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 [8, §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 [8, §1.1.1], symmetric runlength-limited (SRLL) constraints [4], and the charge constraints [8, §1.1.2]. 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.

1) At stage $t = 0, 1, 2, \cdots$, 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 m and $\mathsf{a} \ge 0$ (and $\mathsf{m} + \mathsf{a} + 1 > 0$) 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 [5], 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 [5] 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 [5] 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 [5] 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 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

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

* This work was supported by grant No. 2002197 from the United-States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

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

II. Two-dimensional constraints

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

The concept of a 1-D constraint can formally be generalized to two dimensions (see [5, §1]). Examples of 2-D constraints are 2-D RLL constraints [7], 2-D SRLL constraints [4], and the so-called square constraint [9]. Let $\mathbb{S}$ be a given 2-D constraint over a finite alphabet Σ. We denote by $\mathbb{S}[\ell, w]$ the set of all $\ell \times w$ arrays in $\mathbb{S}$. The capacity of $\mathbb{S}$ [7, Appendix] is given by

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

Suppose we wish to encode information to an $\ell \times w$ array which must satisfy the constraint $\mathbb{S}$; namely, the array must be an element of $\mathbb{S}[\ell, w]$. As a concrete example, consider the square constraint [9]: its elements are all the binary arrays in which an entry may equal '1' only if all its eight neighbors are '0'.

We first partition our array into two alternating types of vertical strips: *data strips* having width w_d and *merging strips* having width w_m. In our example, let $w_\mathrm{d} = 4$ and $w_\mathrm{m} = 1$ (see Figure 1).

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

Fig. 1. Binary array satisfying the square constraint, partitioned into data strips of width $w_\mathrm{d} = 4$ and merging strips of width $w_\mathrm{m} = 1$.

Secondly, we select a graph $G = (V, E, L)$ with a labeling $L : E \to \mathbb{S}[1, w_\mathrm{d}]$ such that $S(G) \subseteq \mathbb{S}$, i.e., each path of length ℓ in G generates a (column) word which is in $\mathbb{S}[\ell, w_\mathrm{d}]$. We then fill up the data strips of our $\ell \times w$ array with $\ell \times w_\mathrm{d}$ arrays corresponding to paths of length ℓ in G. Thirdly, we assume that the choice of w_m allows us to fill up the merging strips in a row-by-row (causal) manner, such that our $\ell \times w$ array is in $\mathbb{S}$. Any 2-D constraint $\mathbb{S}$ for which such w_d, w_m, 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 [4]).

Consider again the square constraint: a graph which produces *all* $\ell \times w_\mathrm{d}$ arrays that satisfy this constraint is given in Figure 2. Also, for $w_\mathrm{m} = 1$, 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 [4].)

Suppose we have an $(\mathsf{m}, 0)$-SBD parallel encoder for $S = S(G)$ at rate R with $M = (w + w_\mathrm{m})/(w_\mathrm{d} + w_\mathrm{m})$ tracks. We may use this parallel encoder to encode information in a row-by-row fashion to our $\ell \times w$ 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.

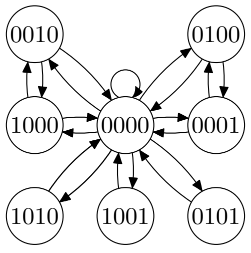

Fig. 2. Graph G whose paths generate all $\ell \times 4$ arrays satisfying the square constraint. The label of an edge is given by the label of the vertex it exits.

The rate at which we encode information to the array is

$$\frac{R}{w_\mathrm{d} + w_\mathrm{m}(1 - 1/M)} \le \frac{\mathsf{cap}(S(G))}{w_\mathrm{d} + w_\mathrm{m}(1 - 1/M)} \ .$$

Taking larger values of w_d (while keeping w_m constant) will typically improve the right-hand side of this inequality. For example, if G is such that it produces all arrays with w_d columns in a given 2-D constraint $\mathbb{S}$, then $\mathsf{cap}(S(G)) \ge w_\mathrm{d} \cdot \mathsf{cap}(\mathbb{S})$. We point however, that typically, the number of vertices and edges in G will grow exponentially with w_d; so, we must take w_d to be reasonably small.

III. Description of the encoder

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

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

$$\mathbf{1} \cdot D = \mathbf{1} \cdot D^T \ , \quad \text{and} \tag{2}$$

$$d_{i,j} > 0 \text{ only if } a_{i,j} > 0 \ . \tag{3}$$

(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.

The N words $\boldsymbol{\gamma}_k = (g_k^{(t)})_{t=1}^{\ell}$, $1 \le k \le N$, 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 product* 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 product of V; that is,

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

An edge $\boldsymbol{e} = \langle e_1, e_2, \dots, e_N\rangle \in E^N$ goes from $\boldsymbol{v} = \langle v_1, v_2, \dots, v_N\rangle \in V^N$ to $\boldsymbol{v}' = \langle v'_1, v'_2, \dots, v'_N\rangle \in V^N$ and is labeled $L^N(\boldsymbol{e}) = \langle b_1, b_2, \dots b_N\rangle$ whenever for all $1 \le k \le N$, e_k is an edge from v_k to v'_k labeled b_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 $S(G^{\otimes N})$.

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

$$\Delta = \frac{\prod_{i\in V} r_i!}{\prod_{i,j\in V} d_{i,j}! \cdot a_{i,j}^{-d_{i,j}}} \tag{4}$$

(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 (4) 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 (2), $\boldsymbol{v}^{(1)}$ 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 [3]. Specifically, suppose that for $0 \le \delta \le n$, an algorithm for encoding information by an n-bit binary vector with Hamming weight δ were given. Suppose also that $V = \{1, 2, \dots, |V|\}$. We could use this algorithm as follows. First, for $n = r_1$ and $\delta = d_{1,1}$, the binary word given as output by the algorithm 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 algorithm 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 enumerative coder 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 V. The above-mentioned preliminary step makes use of the Schönhage–Strassen integer-multiplication algorithm [1, §7.5], and the resulting encoding time complexity is proportional to $M \log^2 M \log\log M$. It turns out that this is also the decoding time complexity. We omit further details due to space limitations.

The next section 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

Throughout this section, we assume a probability distribution on the edges of G, which is the maxentropic stationary Markov chain $\mathcal{P}$ on G [8]. Without real 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, $\boldsymbol{\pi} Q = \boldsymbol{\pi}$. Let

$$M' = M - \lfloor |V| \operatorname{diam}(G)/2 \rfloor \ , \tag{5}$$

and define

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

Note that $\boldsymbol{\rho} = \mathbf{1} \cdot (P)^T$ and $M' = \mathbf{1} \cdot P \cdot \mathbf{1}^T$. Also, observe that (1)–(3) 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, 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 non-integers.

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

$$M' = \textstyle\sum_{i,j\in V} p_{i,j} \quad = \quad \sum_{i,j\in V} \tilde{p}_{i,j} \ , \tag{6}$$

$$\left\lfloor \textstyle\sum_{j\in V} p_{i,j} \right\rfloor \ \le \ \sum_{j\in V} \tilde{p}_{i,j} \ \le \ \left\lceil \sum_{j\in V} p_{i,j} \right\rceil \ , \tag{7}$$

$$\lfloor p_{i,j} \rfloor \ \le \ \tilde{p}_{i,j} \ \le \ \lceil p_{i,j} \rceil \ , \quad \text{and—} \tag{8}$$

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

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 (6)–(9) as an integer flow problem (see Figure 3). Consider the following flow network, with upper and lower bounds on the flow through the edges [2, §6.7]. The network has the vertex set

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

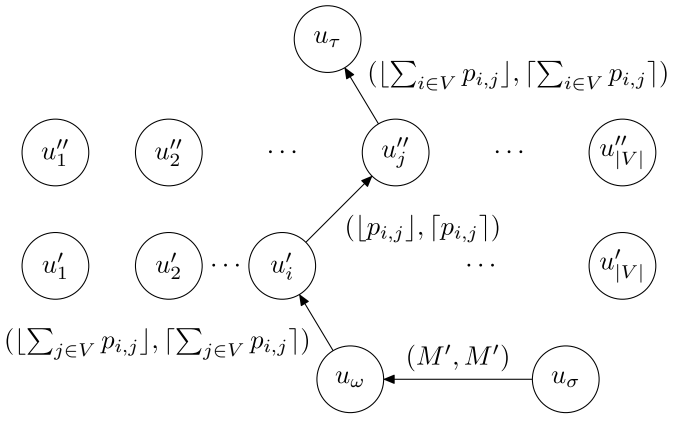
u_τ
$(\lfloor\sum_{i\in V} p_{i,j}\rfloor, \lceil\sum_{i\in V} p_{i,j}\rceil)$
u''_1
u''_2
$\cdots$
u''_j
$\cdots$
$u''_{|V|}$
$(\lfloor p_{i,j}\rfloor, \lceil p_{i,j}\rceil)$
u'_1
u'_2
$\cdots$
u'_i
$\cdots$
$u'_{|V|}$
$(\lfloor\sum_{j\in V} p_{i,j}\rfloor, \lceil\sum_{j\in V} p_{i,j}\rceil)$
u_ω
(M', M')
u_σ

Fig. 3. Flow network for the proof of Lemma 2.

with source u_σ and target u_τ. Henceforth, when we talk about 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 equaling to M'.
2) $u_\omega \to u'_i$ for every $i \in V$, with the upper and lower bounds $\lfloor\sum_{j\in V} p_{i,j}\rfloor$ and $\lceil\sum_{j\in V} p_{i,j}\rceil$, 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.
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 (6)–(9) 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 [2, Theorem 6.5]. Moreover, such a flow can be efficiently found [2, §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 $u_\sigma \to u'_i$ 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).

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,

$$||\tilde{\boldsymbol{r}} - \tilde{\boldsymbol{\rho}}|| \le |V| \ ,$$

where $||\cdot||$ denotes the $\mathcal{L}_1$ norm of a real vector.

Proof: From (7), we get that for all $i \in V$,

$$\lfloor\textstyle\sum_{j\in V} p_{i,j}\rfloor \le \tilde{\rho}_i \le \lceil\sum_{j\in V} p_{i,j}\rceil \ . \qquad (10)$$

Recall that (2) is satisfied if we replace D by P. Thus, by (9), we have that (10) also holds if we replace $\tilde{\rho}_i$ by $\tilde{r}_i$. We conclude that $|\tilde{\rho}_i - \tilde{r}_i| \le 1$. ■

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 4: 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) $d_{i,j} \ge \tilde{p}_{i,j}$ for all $i, j \in V$, and—
2) $M' \le \mathbf{1}\cdot D\cdot\mathbf{1}^T \le M$

(where M' is as defined in (5)). Moreover, the matrix D can be found by an efficient algorithm.

Partial proof: Consider a vertex $i \in V$. If $\tilde{r}_i > \tilde{\rho}_i$, then we say that vertex i has a *surplus* of $\tilde{r}_i - \tilde{\rho}_i$. On the other hand, if $\tilde{r}_i < \tilde{\rho}_i$ then vertex i has a *deficiency* of $\tilde{\rho}_i - \tilde{r}_i$. 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:

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

Our aim is to use the surplus in order to eventually make up for the deficiency. We next show how this can be done.

Given $\tilde{\boldsymbol{\rho}}$ and $\tilde{\boldsymbol{r}}$, we build for $\theta = \mathrm{diam}(G)$ the following flow network. The vertex set of the network is given by

$$\{u_\sigma, u_\tau\} \cup \{u_{h,i} \ : \ 1 \le h \le \theta+2 \quad , \ i \in V\} \ ,$$

where u_σ is the source and u_τ is the sink. The edge set E' consists of the following edges:

1) Surplus edges: For $i \in V$,
$$e' = u_\sigma \to u_{1,i} \in E' \quad \text{iff} \quad \tilde{r}_i > \tilde{\rho}_i \ ,$$
with the upper and lower bounds on e' both equaling $\tilde{r}_i - \tilde{\rho}_i$.
2) Trellis edges: For $1 \le h < \theta + 1$ and $i, j \in V$,
$$e' = u_{h,i} \to u_{h+1,j} \in E' \quad \text{iff} \quad e = i \to j \in E \ ,$$
with upper and lower bounds ∞ and 0, respectively.
3) Identity edges: For $i, i' \in V$ and $1 \le h \le \theta + 1$,
$$e' = u_{h,i'} \to u_{\theta+2,i} \in E' \quad \text{iff } i' = i \ ,$$
with upper and lower bounds ∞ and 0, respectively.
4) Deficiency edges: For $i \in V$,
$$e' = u_{\theta+2,i} \to u_\tau \in E' \quad \text{iff} \quad \tilde{r}_i < \tilde{\rho}_i \ ,$$
with the upper and lower bounds both equaling $\tilde{\rho}_i - \tilde{r}_i$.

Let $\varphi : E' \to \mathbb{Z}$ be a legal integer flow in our flow network (we omit the details of why such a flow indeed exists). Define the $|V| \times |V|$ matrix $D = (d_{i,j})$ by

$$d_{i,j} = \tilde{p}_{i,j} + \sum_{h=1}^{\theta}\varphi(u_{h,i} \to u_{h+1,j}) \ .$$

This definition already implies that D satisfies (3), as well as part 1 of the theorem. We next establish (2). On the one hand,

$$\begin{aligned}(\mathbf{1}\cdot(D)^T)_i &= \sum_{j\in V}\tilde{p}_{i,j} + \sum_{j\in V}\sum_{h=1}^{\theta}\varphi(u_{h,i} \to u_{h+1,j}) \\ &= \tilde{\rho}_i + \sum_{j\in V}\sum_{h=1}^{\theta}\varphi(u_{h,i} \to u_{h+1,j}) \ . \qquad (11)\end{aligned}$$

On the other hand, denoting by $\varphi(u)$ the total flow entering (and leaving) vertex u, we have

$$\begin{aligned}(\mathbf{1}\cdot D)_i &= \sum_{k\in V} d_{k,i} = \sum_{k\in V}\tilde{p}_{k,i} + \sum_{k\in V}\sum_{h=1}^{\theta}\varphi(u_{h,k}\to u_{h+1,i})\\ &= \tilde{r}_i + \sum_{h=1}^{\theta}\varphi(u_{h+1,i}) = \tilde{r}_i + \sum_{h=2}^{\theta+1}\varphi(u_{h,i})\\ &= \tilde{r}_i - \varphi(u_{1,i}) + \sum_{h=1}^{\theta+1}\varphi(u_{h,i})\\ &= \underbrace{\tilde{r}_i - \varphi(u_{1,i}) + \varphi(u_{\theta+2,i})}_{\tilde{\rho}_i} + \sum_{j\in V}\sum_{h=1}^{\theta}\varphi(u_{h,i}\to u_{h+1,j}) \ .\end{aligned}$$

Hence, $\mathbf{1}\cdot(D)^T = (\mathbf{1}\cdot D)$.

The first inequality in part 2 of the theorem follows from the fact that $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i,j\in V$; specifically,

$$M' = \mathbf{1}\cdot P\cdot\mathbf{1}^T \overset{(7)}{=} \mathbf{1}\cdot\tilde{P}\cdot\mathbf{1}^T \leq \mathbf{1}\cdot D\cdot\mathbf{1}^T \ .$$

Next we turn to the second inequality in part 2. By (11),

$$\mathbf{1}\cdot D\cdot\mathbf{1}^T = M' + \textstyle\sum_{h=1}^{\theta}\sum_{i\in V}\varphi(u_{h,i}) \ .$$

From the structure of the flow network we get that for $1 \leq h \leq \theta$,

$$\sum_{i\in V}\varphi(u_{h,i}) \leq \varphi(u_\sigma) = \mathrm{Surp} = \tfrac{1}{2}||\tilde{\boldsymbol{r}} - \tilde{\boldsymbol{\rho}}|| \leq \tfrac{1}{2}\,|V| \ ,$$

where the last inequality follows from Lemma 3. Thus,

$$\mathbf{1}\cdot D\cdot\mathbf{1}^T \leq M' + \theta\cdot\mathrm{Surp} \leq M' + |V|\,\mathrm{diam}(G)/2 = M \ .$$

Observe that part 2 implies (1). ■

Proof sketch of Theorem 1: Let $\tilde{\Delta}$ be as in (4), where we replace $d_{i,j}$ by $\tilde{p}_{i,j}$ and r_i by $\tilde{\rho}_i$. Since $d_{i,j} \geq \tilde{p}_{i,j}$ for all $i,j\in V$, we have

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

We may now bound $\lfloor\log_2\tilde{\Delta}\rfloor/M'$ from below using Stirling's approximation. The proof ends by taking into account the various quantization errors introduced into $\tilde{P}$. ■

V. Fast enumerative coding

Recall from Section III that in the course of our encoding algorithm, we encode information into fixed-length binary words of constant weight. A way to do this would be to use enumerative coding [3]. Immink [6] 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 must encode into. Assume our variables are floating-point numbers with a mantissa of μ bits and an exponent of ϵ bits: each floating-point number is of the form $\overline{x} = a\cdot 2^b$ where a is an integer such that $0 \leq a - 2^\mu < 2^\mu$ and b is an integer such that $-2^{\epsilon-1} \leq b < 2^{\epsilon-1}$. Note that $\mu+\epsilon$ bits are needed to store such a number.

We assume the presence of two look-up 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

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

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 look-up 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 $0 \leq \chi \leq \kappa \leq n$ we define

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

Note that since we have stored the relevant numbers in our look-up table, the calculation of the above function takes only $O(\mu^2+\epsilon)$ time. The encoding algorithm is given in Figure 4, and is clearly invertible. Since we must take $\epsilon = O(\log n)$ and $\mu = O(\log\delta)$, its running time is $O(n\log^2 n)$.

Name: EnumEncode(n,δ,ψ)

Input: Integers n,δ,ψ such that $0\leq\delta\leq n$ and $0\leq\psi<\left[\begin{smallmatrix}n\\ \delta\end{smallmatrix}\right]$.
Output: A binary word of length n and weight δ.

```
if ($\delta$ == 0) // stopping condition:
    return $\underbrace{00\ldots0}_{n}$;
for ($\iota \leftarrow 1$; $\iota \leq n-\delta+1$; $\iota$++) {
    if ($\psi < \left[\begin{smallmatrix}n-\iota\\ \delta-1\end{smallmatrix}\right]$)
        return $\underbrace{00..0}_{\iota-1}$1||EnumEncode($n-\iota, \delta-1, \psi$);
    else
        $\psi \leftarrow \psi - \left[\begin{smallmatrix}n-\iota\\ \delta-1\end{smallmatrix}\right]$;
}
```

Fig. 4. Enumerative encoding algorithm for constant-weight binary words.

References

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

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*. Engelwood Cliffs, New Jersey: Prentice Hall, 1993.

[3] T. Cover, "Enumerative source coding," *IEEE Trans. Inform. Theory*, 19 (1973), 73–77.

[4] T. Etzion, "Cascading methods for runlength-limited arrays," *IEEE Trans. Inform. Theory*, 43 (1997), 319–324.

[5] S. Halevy and R. M. Roth, "Parallel constrained coding with application to two-dimensional constraints," *IEEE Trans. Inform. Theory*, 48 (2002), 1009–1020.

[6] K. A. S. Immink, "A practical method for approaching the channel capacity of constrained channels," *IEEE Trans. Inform. Theory*, 43 (1997), 1389–1399.

[7] A. Kato and K. Zeger, "On the capacity of two-dimensional run-length constrained code," *IEEE Trans. Inform. Theory*, 45 (1999), 1527–1540.

[8] 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: Elsevier, 1998, pp. 1635–1764.

[9] W. Weeks and R. E. Blahut, "The capacity and coding gain of certain checkerboard codes," *IEEE Trans. Inform. Theory*, 44 (1998), 1193–1203.