Variable-Length State Splitting with Applications to Average Runlength-Constrained (ARC) Codes

Chris D. Heegard, *Member*, *IEEE*, Brian H. Marcus, *Member*, *IEEE*, and Paul H. Siegel, *Member*, *IEEE*

***Abstract*—A new class of constrained systems: average runlength constraints (ARC) are defined. These systems are defined by requiring that the sum of n consecutive runlengths be bounded above by a linear function of n. In particular, the *running average runlength* of every sequence in the system is bounded above by a constant. A general result is given on the capacity of ARC systems. The state splitting algorithm is then improved for variable-length graphs. This is then applied to obtain high, fixed-rate codes from the free binary source to ARC systems. As an example, a rate $1/2$, $(d,k)=(2,7)$ code is constructed that has a smaller average runlength than the industry standard $(2,7)$ code.**

***Index Terms*—Magnetic recording, runlength-limited codes, state-splitting.**

I. Introduction

THE CONSTRUCTION of efficient codes for input-constrained channels has been a subject of research since Shannon's classical investigation of discrete noiseless channels [1]. The sliding-block code algorithm of Adler, Coppersmith, and Hassner [2], which built upon earlier work of Franaszek [3], [4], Patel [5], and Marcus [6], provided a systematic and mathematically rigorous approach to designing practical codes for finite-memory, constrained systems. These sliding-block codes are characterized by finite-state, fixed-rate encoders and sliding-block (that is, state-independent) decoders. Among the constrained systems to which the algorithm applies is the important class of runlength-limited (RRL) constraints that have been predominant in digital magnetic storage for over three decades.

Input-constrained channels are often represented by finite-state transition-diagrams (FSTD) where the edge labels of the underlying directed graph consist of a fixed number of code symbols. Certain constraints, including the RLL constraints, can be represented more compactly if one utilizes a variable-length graph (VLG), where the edge labels in the FSTD may have different lengths, as in Shannon's original description of the telegraph channel. The state-splitting technique, central to the algorithm in [2], was extended by Adler, Friedman, Kitchens, and Marcus [7] to constraints described by a VLG. This generalized approach proved to be most effective in the construction of fixed-rate $(1:1)$ codes from N-ary data to the constrained system; the application to the construction of codes with arbitrary rate $(p:q)$ often requires representing the constrained sequences in a way that is not consistent with the original variable-length structure.

In this paper, we develop an improved variable-length state-splitting algorithm that permits the design of rate $(p:q)$ codes more directly from the original VLG representation of the constraint. The modified algorithm is then applied to a new class of constraints—called *average runlength constrained* (*ARC*) *systems*—that represent a natural generalization of the familiar RLL (d,k) constraints. These constraints place a nontrivial restriction upon the average runlength of binary sequences satisfying a specified (d,k) constraint. The reduction of the average runlength is equivalent to an increased average density of 1's in the code sequences. Since most timing recovery and gain control algorithms in digital recording systems using peak detection are data-driven, the larger density of peaks in the readback signal (corresponding to the 1's in the recorded ARC sequence) can translate into improved performance of these control loops.

The outline of the remainder of the paper is as follows.

In Section II, we introduce the ARC systems, and describe simple representations in terms of VLG's. Several results related to the Shannon capacity and statistics of the ARC sequences are then derived. We also point out that the problem of computing capacities of ARC systems can be viewed from the perspective of costly constrained channels or the theory of large deviations.

In Section III, we develop the generalized techniques for construction of efficient fixed-rate, variable-length codes, and we indicate how the new methods can shorten the construction procedure in [7].

Manuscript received August 16, 1990; revised March 4, 1991. This work was supported in part by NSF Grants ECS83-52220 and NCR89-03931. This work was presented in part at the IEEE International Symposium on Information Theory, Kobe, Japan, June 1988.

C. D. Heegard is with the School of Electrical Engineering, 327 Engineering Theory Center, Cornell University, Ithaca, NY 14853.

B. H. Marcus and P. H. Siegel are with IBM Research Division, Almaden Research Center, K61/802, 650 Harry Road, San Jose, CA 95120.

IEEE Log Number 9144528.

In Section IV, we describe two applications of the new variable-length code construction methods developed in Section III. In the first example, the well-known rate 1/2, variable-length, $(d,k)=(2,7)$ RLL code invented by Franaszek is rederived using the variable-length state-splitting approach. For the second example, we consider an ARC system satisfying the same runlength constraints, but having an average runlength strictly smaller than the conventional (2,7) constraint (or, in fact, the industry-standard, rate 1/2 (2,7) code). We then describe the design of a rate 1:2, variable-length code into this ARC system.

The Appendix to the paper contains proof details for several of the results developed in Sections II and III.

II. Average Runlength Constrained (ARC) Codes

A. The (d, k) Runlength Constraint

A runlength constrained code $\mathscr{C}$ is often described by a set of *transition* sequences. Each transition sequence (or string) in the code is a binary sequence $x=x_1,x_2,\cdots,x_n$, $x_i\in\{0,1\}$, that satisfies certain runlength parameters. For example, a (d,k) constrained sequence x has the property that the length of all substrings of "0"s between consecutive "1"s is at least d and at most k [8]. That is, if

$$x_{i-1}=1, \qquad x_i=x_{i+1}=\cdots=x_{i+m-1}=0, \qquad x_{i+m}=1,$$

then $d\le m\le k$. This constraint is often described by a fixed-length graph with $k+1$ vertices (if k is finite) or $d+1$ vertices (if $k=\infty$). See Fig. 1.

The graph is termed *fixed-length* since the length of each edge of the graph is a constant; in this case each edge represents one bit of the transition sequence.

The *base-b capacity* of a constraint, such as the (d,k) runlength constraint, is equal to the limit, as $n\to\infty$, of $1/n$ times the base-b logarithm of the number of strings of length n that satisfy the constraint. (In this paper all logarithms are to the base 2). The capacity is always the logarithm of the largest root of a polynomial equation, and there are several methods for computing it for a given constraint [2], [8]–[10]. In the case of the (d,k) constraint, the capacity is equal to $\log(\lambda)$ where λ is the largest root of

$$\lambda^{k+1}=1+\lambda+\cdots+\lambda^{k-d}, \qquad \text{for finite } k \tag{1a}$$

and

$$\lambda^{d+1}=1+\lambda^{d}, \qquad \text{for } k=\infty. \tag{1b}$$

Another method of describing runlength constraints, that is often quite useful, is in terms of the runlengths themselves. As will be demonstrated, these descriptions are inherently presented by a variable-length graph. For a given transition sequence x, define the *transition times*

$$t_i=\min\{j>t_{i-1}|x_j=1\},$$

(where $t_0\equiv 0$). Then the *runlengths* are defined as the differences $T_i=t_i-t_{i-1}$. For a sequence that satisfies a (d,k) runlength constraint, the runlengths must satisfy the bounds $d+1\le T_i\le k+1$ for all i. This is in fact an equivalent description of the (d,k) constraint. The variable-length graph for this constraint has a single vertex (see Fig. 2). The graph is termed *variable-length* since the lengths of the edges in the graph are not all equal.

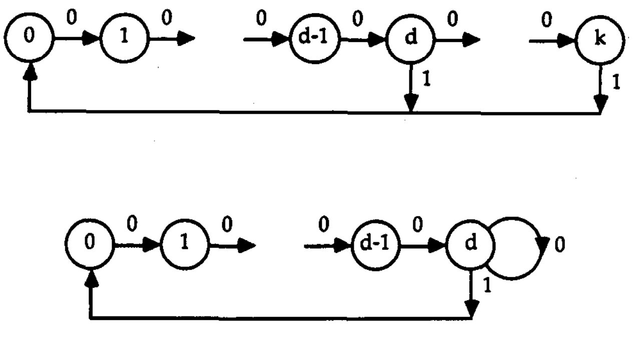

Fig. 1. Fixed-length (d,k) graph.

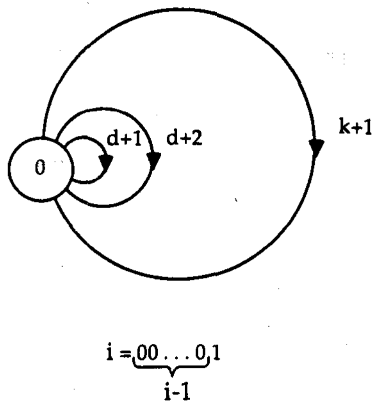

Fig. 2. Variable-length (d,k) graph.

B. The (d, k, a, b) Average Runlength Constraint (ARC)

The *average runlength* $\overline{T}$ of a runlength sequence, $T=T_1,T_2,\cdots,T_n$, is defined by the average

$$\overline{T}=\frac{1}{n}\sum_{i=1}^{n}T_i. \tag{2}$$

It is of interest to describe sets of sequences that maintain a uniform upper bound a on the average runlength. As an example, the (d,k) constraint maintains a bound $\overline{T}\le k+1$ on every code sequence. A nontrivial average runlength constraint satisfies $\overline{T}\le a<k+1$ for every sequence as $n\to\infty$. Such a constraint on the average runlength will guarantee a minimum density $1/a$ of "1"s in the transition sequence. Thus, an ARC constraint can potentially provide for improved timing recovery beyond that guaranteed by the k constraint itself.

An *average runlength constrained* (ARC) code is described by four parameters (d,k,a,b). In such a system, the runlengths satisfy $d+1\le T_i\le k+1$ and for all $1\le m\le n$

$$\frac{1}{n-m+1}\sum_{i=m}^{n}T_i\le a+\frac{b}{n-m+1}$$

or equivalently

$$\sum_{i=m}^{n} (T_i - a) \le b. \tag{2'}$$

Note that this constraint is stationary and implies a bound of $a + b/n$ on the average runlength, (2); in the limit as $n \to \infty$, the average runlength is bounded by a. Conversely, if runlength sequences are generated by a finite state machine (e.g., an encoder) and for a given a, in the limit as $n \to \infty$, the average runlength (2) is uniformly bounded by a, then (2) is bounded above by $a + b/n$ for some b. To see this, first observe that every runlength sequence that is generated by a *cycle* (of n consecutive edges) in the finite state machine must satisfy

$$\frac{1}{n}\sum_{i=1}^{n} T_i \le a,$$

for some value of a. Then observe that, for every runlength sequence, the cumulative sum of runlengths equals the sum of runlengths of a set of cycles and a sequence of uniformly bounded length. The "best" value for a is the maximum average runlength of all *simple* (i.e., non-self-intersecting) cycles (there are only finitely many), and the "best" value for b is determined by maximizing

$$\sum_{i=1}^{n} (T_i - a)$$

over the simple paths of the finite state machine.

The ARC, for a and b integers, is conveniently described in terms of a graph with $b+1$ vertices. Label the vertices of the graph with the numbers $0, 1, \cdots, b$. Then from each vertex $0 \le i \le b$, draw an edge to vertex $0 \le j \le b$, labeled with a runlength of length l, if $d+1 \le l \le k+1$ and

$$j = \max\{0, i + l - a\}.$$

To see this, argue as follows. Suppose that $T_1 \cdots T_n$ satisfies the ARC constraint (2'). Inductively, define $S_0 = 0$ and

$$S_i \equiv \max\{0, S_{i-1} + T_i - a\}.$$

It suffices to show that whenever $S_i > 0$ then $S_i \le b$. For then, the sequence $S_1 \cdots S_n$ is the state sequence of a path in the ARC graph that generates $T_1 \cdots T_n$. For each such i, let $i_0 = \max\{j < i: S_j = 0\}$. Then

$$S_i = \sum_{j=i_0+1}^{i} (T_j - a) \le b.$$

Conversely, every sequence of runlengths $T_1 \cdots T_n$ generated by the ARC graph satisfies the ARC because if $S_0 \cdots S_n$ is the corresponding state sequence, then

$$\sum_{i=1}^{n} (T_i - a) \le S_n \le b.$$

As an example, consider the constraint $(d,k,a,b) = (1,7,6,2)$ as shown in Fig. 3.

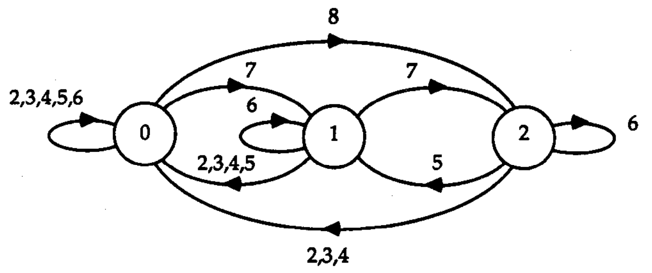

Fig. 3. $(d,k,a,b) = (1,7,6,2)$ ARC constraint.

C. *The ARC Capacity*

The capacity of the ARC can be determined from the $(b+1)\times(b+1)$ adjacency (or transition) matrix $A(D)$ that describes the graph. The components of the matrix, $a_{i,j}(D)$, $0 \le i, j \le b$, are polynomials in the variable D. If an edge appears in the graph from vertex i to vertex j with length l, then the monomial D^l appears as a term in the polynomial $a_{i,j}(D)$. For example, for the constraint $(d,k,a,b) = (1,7,6,2)$,

$$A(D) = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & D^2 + D^3 + D^4 + D^5 + D^6 & D^7 & D^8 \\ 1 & D^2 + D^3 + D^4 + D^5 & D^6 & D^7 \\ 2 & D^2 + D^3 + D^4 & D^5 & D^6 \end{array}.$$

The capacity is then equal to $\log(\lambda)$ where λ is the largest root of $\det(I - A(D^{-1}))$ [1].

For small values of b, this determinant is given by:

$b = 0$ $(d+1 \le a,\ k+1 = a)$

$$\det(I - A(D)) = 1 - D^{d+1} - D^{d+2} - \cdots - D^a$$

$b = 1$ $(d+1 \le a,\ k+1 = a+1)$

$$\det(I - A(D)) = 1 - D^{d+1} - D^{d+2} - \cdots - D^{a-1} - 2D^a + D^{a+d+1}$$

$b = 2$ $(d+1 \le a,\ k+1 = a+1)$

$$\det(I - A(D)) = 1 - D^{d+1} - D^{d+2} - \cdots - D^{a-1} - 3D^a + 2D^{a+d+1} + D^{a+d+2} + D^{a+d+3} \cdots + D^{2a} + D^{2a+d+2}$$

$b = 2$ $(d+1 \le a,\ k+1 = a+2)$

$$\det(I - A(D)) = 1 - D^{d+1} - D^{d+2} - \cdots - D^{a-1} - 3D^a + 2D^{a+d+1} + D^{a+d+2}.$$

For example, the capacity of $(d,k,a,b) = (1,7,6,2)$ is $0.6783 \cdots > \frac{2}{3}$ bits. This means that it is possible to code the free binary source at a rate of $\frac{2}{3}$ bits and maintain these constraints. It is interesting to note that the rate $\frac{2}{3}$, $(d,k) = (1,7)$ code [11] satisfies the $(d,k,a,b) = (1,7,6,2)$ constraints. The popular $(2,7)$ code [11] does not satisfy a non-trivial "a" constraint; it is a $(d,k,a,b) = (2,7,8,0)$, rate $\frac{1}{2}$ code. However, the capacity of the ARC with $(d,k,a,b) = (2,7,6,3)$ is $0.5128 \cdots$ bits; thus it is possible to construct a rate $\frac{1}{2}$ code satisfying this nontrivial constraint. Such a code is derived in Section IV.

D. The ARC Capacity — Large b

The capacity of the ARC for large values of b is of particular interest since for any finite b, the average runlength (2) is bounded by a in the limit as $n \to \infty$. Of course, for large values of b, the short term average runlength can be much larger than a and the complexity of the constraint grows (recall that the ARC graph has $b+1$ vertices).

Let λ^* be the solution to (1), which determines the capacity of the (d,k) constraint. Define

$$a^* = \sum_{j=d+1}^{k+1} j(\lambda^*)^{-j}.$$

Note that a^* is equal to the average runlength of the ensemble of the set of runlength sequences that satisfy the (d,k) constraint. In the limit as $b \to \infty$ there are two cases. The two cases depend upon the relationship between the constraint parameter a and the value of a^*.

Theorem 1:

a) If $a \geq a^*$, then the limiting capacity of the (d,k,a,b) ARC as $b \to \infty$ is equal to

$$\log(\lambda^*).$$

b) If $d+1 < a < a^*$ the limiting capacity as $b \to \infty$ is equal to

$$\left(1-\frac{d}{a}\right)\log(\lambda)+\frac{1}{a}\log\left(\frac{1-\lambda^{d-k-1}}{\lambda-1}\right),$$

where λ is the unique positive solution to the equation

$$a(1+\lambda+\cdots+\lambda^{k-d}) = (k+1)+k\lambda+\cdots+(d+1)\lambda^{k-d},$$

if k is finite and

$$\lambda = \frac{a-d}{a-d-1},$$

when $k=\infty$.

Note that when the maximum runlength is unbounded, $k=\infty$, and $d+1<a<a^*$, the capacity is equal to

$$\left(1-\frac{d}{a}\right)h\left(\frac{1}{a-d}\right)$$

in the limit as $b \to \infty$, where $h(p)$ is the binary entropy function given by

$$h(p) = -p\log(p)-(1-p)\log(1-p).$$

With some interpretation, one can derive Theorem 1 as a special case of formula (28) of Csiszar, Cover, and Choi [12] (c.f. also Justesen and Hoholt [13] and Karabed, Khayrallah, and Neuhoff [14]). For completness, we give a direct proof in the Appendix. The rough idea is as follows. Using the ideas of [9], Theorem 1 has a simple interpretation in terms of an entropy maximization problem. Let $T \in \{d+1, d+2, \cdots, k+1\}$ be a random variable with distribution $p_j = \Pr(T=j)$. Consider the problem of maximizing the entropy of the random variable T divided by its mean, $H(T)/E(T)$. This quantity (which is measured in bits of information per unit time) bounds the rate of any (d,k) encoder. The optimal distribution for this maximization is geometric, $p_j = (\lambda^*)^{-j}$, $H(T)/E(T) = \log(\lambda^*)$ and $E(T)=a^*$. This is Case a) of Theorem 1, and $H(T)/E(T)$ is the limiting capacity. If a further constraint is imposed, namely $E(T) \leq a$, then it is clear that if $a \geq a^*$, this does not affect the optimal distribution. On the other hand, if $a<a^*$, the additional constraint is nontrivial. In this case, the optimal distribution is again geometric with $p_j = \lambda^{-j}/\Sigma_i \lambda^{-i}$, $H(T)/E(T) = \log(\lambda)+\log(\Sigma_i \lambda^{-i})/a$ and $E(T)=a$ (λ is determined by the latter.) This is Case b) of Theorem 1, and $H(T)/E(T)$ is again the limiting capacity whyich can be algebraically reduced to the form in the theorem. When $a \geq a^*$, the limiting capacity of the (d,k,a,b) constraint, as $b \to \infty$, is the same as the capacity of the (d,k) constraint. On the other hand, when $d+1<a<a^*$, a loss in capacity is incurred. See Fig. 4.

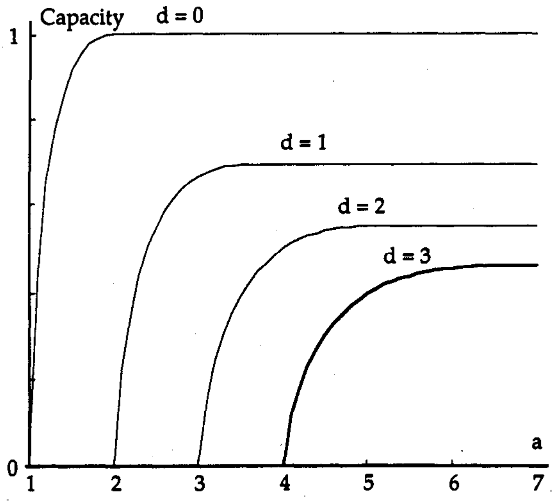

Fig. 4. Capacity versus a for $b=k=\infty$.

III. Variable-Length State Splitting

A. Definitions and Background

The *state-splitting* method for constructing fixed-rate codes from a free (i.e., unconstrained) N-ary source ($N \geq 2$) into a constrained system of symbols is described in [2]. The method depends on a fixed-length presentation of the constraint; in particular, a labeling of a finite directed graph by symbols of constant length. Many constrained systems, such as systems based on runlength constraints, are naturally described by a variable-length labeling. Of course, they can also be represented by fixed-length graphs, but the variable-length representation is often more compact. In [7], the algorithm of [2] was extended to variable-length presentations, yielding simpler constructions of fixed-rate codes. The extension of the algorithm was described for rate (1:1) codes. This method can be applied to give rate $(p:q)$ codes by blocking arbitrary N-ary sequences into p-blocks and the constrained sequences into q-blocks. However, this approach involves finding a presentation of the constraint in q-blocks, thereby destroying the original variable-length presentation. In the present paper, we show how to avoid this by modifying the state-splitting algorithm in [7]. In the course of doing this, we indicate how the algorithm in [7] can be shortened in some cases. The modified variable-length state-splitting algorithm is then used to construct ARC codes.

In order to state the results precisely, we will make the following definitions.

A *(finite, directed) graph* is a pair, $\mathscr{G}=(\mathscr{S},\mathscr{E})$, of finitely many *states* (or *vertices*), $\mathscr{S}$, and finitely many *edges* $\mathscr{E}$. Each edge has a starting state and ending state. We let $\mathscr{E}_{i,j}$ denote the set of edges from state i to state j, and $\mathscr{E}_i$ denote the set of outgoing edges from state i:

$$\mathscr{E}_i=\cup_j\mathscr{E}_{i,j}.$$

We usually assume that the graph $\mathscr{G}$ is *irreducible*: there is a path (i.e., a sequence of edges) from each state to every other state.

A *labeled graph* is a triple $(\mathscr{G},\mathscr{W},w)$ where $\mathscr{G}$ is a graph, $\mathscr{W}$ is a finite alphabet and w is a *labeling function*

$$w\colon \mathscr{E}\to\mathscr{W}^*,$$

where $\mathscr{W}^*=\cup_{k=0}^{\infty}\mathscr{W}^k$. The labeling function assigns a word $w(e)=w_1w_2\cdots w_l$ of length $l>0$ to each edge of the graph $\mathscr{G}$. For a labelled VLG, the labelling w on edges determines a labeling $w(\gamma)$ on paths γ. A labeling is called *right-closing* if for each sufficiently long word $w=w_1\cdots w_n$ and each state i, there is an edge $e=e(w,i)$ such that, if γ is a path that begins at state i and if w is a prefix of $w(\gamma)$, then γ begins with e. A labeling is called *right-resolving* if the outgoing edges are labeled distinctly and the labels constitute a prefix-free list. Clearly, every right-resolving labeling is right-closing, but the converse is false. The right-closing property can be thought of as a "delayed" version of right-resolving. Note that the labeling of the RLL and ARC systems in Figs. 2 and 3 are right-resolving. The terms right-closing and right-resolving, introduced in symbolic dynamics, have in the past been applied to labelings of fixed-length graphs. In that context, the term right-closing means the same as the expressions *lossless of finite order* or *of local finite anticipation*, and the term right-resolving means the same as the terms *unifilar* (used in source coding) and *deterministic* (used in automata theory) [15].

A *variable-length graph* (VLG) is a pair, $(\mathscr{G},l)$, consisting of a finite directed graph $\mathscr{G}$, and a *length function*

$$l\colon \mathscr{E}\to\mathbb{Z}^+\equiv\{1,2,\cdots\},$$

defined on the edges of $\mathscr{G}$, which takes on positive integer values. A *fixed-length graph* is a VLG with a length function that is equal to a constant.

A *subgraph* of a VLG $(\mathscr{G},l)$ is a VLG $(\mathscr{G}',l')$ satisfying $\mathscr{S}'\subset\mathscr{S}$, $\mathscr{E}'\subset\mathscr{E}$, $l'(e)=l(e)$ for every edge $e\in\mathscr{E}'$, where $\mathscr{G}=(\mathscr{S},\mathscr{E})$ and $\mathscr{G}'=(\mathscr{S}',\mathscr{E}')$.

A VLG is obtained from a labeling of a graph $\mathscr{G}$ by taking the length of the edge to be the length of the label. VLG's that are obtained from a labeling of a graph are the main concern of this paper. The length function determines the length of the paths in the graph: if

$$\gamma=e_1e_2\cdots e_k$$

is a path through $\mathscr{G}$ (where the $e_i\in\mathscr{E}$), then

$$l(\gamma)\equiv\sum_{i=1}^{k}l(e_i)$$

is the *length* of γ.

A path through the graph,

$$\gamma=e_1e_2\cdots e_k,$$

is said to be a *cycle* of $\mathscr{G}$ if the beginning state of edge e_1 coincides with the ending state of edge e_k.

Let $P(\mathscr{G},l)$, the *period* of an irreducible $(\mathscr{G},l)$, denote the greatest common divisor of the lengths of the cycles of $\mathscr{G}$. If an integer q divides $P(\mathscr{G},l)$, then there exists a *phase function*

$$c\colon\mathscr{S}\to\{0,1,\cdots,q-1\}$$

such that if e is an edge from state i to state j, then

$$c(j)=l(e)+c(i)\text{ modulo }q. \tag{3}$$

Such a function is uniquely determined once $c(i_0)$ is set equal to 0 for some (arbitrary) state i_0. Namely, set $c(i)=l(\gamma)$ modulo q where γ is any path from i_0 to i. The reader may check that $c(i)$ is well defined, independent of the choice of γ, and satisfies (3).

Finally, let $A(D)$ denote the *adjacency matrix* of a VLG $(\mathscr{G},l)$. The i,j component of $A(D)$ is defined to be

$$A_{i,j}(D)\equiv\sum_{e\in\mathscr{E}_{i,j}}D^{l(e)}.$$

Conversely, given a matrix $A(D)$, whose entries are integer polynomials in D, the matrix is an adjacency matrix of a VLG if (1) the coefficients of the polynomials in $A(D)$ are nonnegative and (2) the exponents are strictly positive.

Define $C(\mathscr{G},l)=\log(\lambda)$ as the *capacity* of $(\mathscr{G},l)$ where λ is the largest root of the equation $\det(I-A(D^{-1}))=0$. If a constrained system is defined by a right-closing labeling, then $C(\mathscr{G},l)$ is the capacity of the constraint, as described at the beginning of Section II.

B. Encodability

As in [7], starting with a variable-length graph (satisfying necessary capacity and periodicity assumptions) a sequence of variable-length graphs is produced via a state-splitting algorithm. The splitting ultimately produces a new variable-length graph that is suitable for use as a finite-state encoder in the following sense.

Let N be the input alphabet size, a positive integer, and p,q a relatively prime pair of positive integers. A

VLG $(\mathcal{S},l)$ is *N-codable at rate* $(p:q)$ if there exists a length function, which we will call an *input length function*,

$$l^*: \mathcal{E} \to \mathbb{Z}^+ \tag{4a}$$

such that the rate is constant on cycles: for every cycle γ

$$l^*(\gamma) = \frac{p}{q} l(\gamma), \tag{4b}$$

and the Kraft Inequality is satisfied with equality for all $i \in \mathcal{S}$,

$$\sum_{e \in \mathcal{E}_i} N^{-l^*(e)} = 1. \tag{5}$$

(This differs slightly from the definition in [7]).

Suppose a VLG, $(\mathcal{S},l)$, is N-codable at rate $(p:q)$ and is obtained from a labeled graph, $(\mathcal{S},\mathcal{W},w)$, that is right closing. Then it is possible to define an encoder that maps the free N-ary source into the constraint described by the labeled graph, at rate $(p:q)$. The encoder is constructed as follows. By (5), we can apply the Kraft Theorem (see [19, p. 41]) to construct, for each state i of $\mathcal{S}$, a complete, prefix-free list of N-ary words $a(e) \in \mathcal{W}^*$, with lengths $l^*(e)$ for each edge $e \in \mathcal{E}_i$. (Complete means that every N-ary word either is a prefix of a word in the list or has a prefix in the list). The encoder is then defined by assigning the input word $a(e)$ to the output word $w(e)$. The encoder, as defined, has rate $(p:q)$ on every cycle, by (4). (It can be shown that the encoder can be transformed to a rate $(p:q)$ encoder in the standard sense. For an example, see the discussion at the end of Section IV-B.) The right-closing property of the labeling implies that the encoder has an inverse—a (state-dependent) decoder.

Let p,q be positive integers and suppose that q divides the period of a VLG $(\mathcal{S},l)$. For each state i, let $m_i \geq 0$ be a nonnegative integer, and let $V(D)$ be the diagonal matrix with entries

$$V_{i,i}(D) = D^{m_i + c(i)p/q}, \tag{6}$$

where $c(i)$ is the phase function described by (3). The matrix

$$A^*(D) = V(D)A(D^{p/q})V(D^{-1}) \tag{7a}$$

is called an *input matrix*. Note that

$$A^*_{i,j}(D) = \sum_{e \in \mathcal{E}_{i,j}} D^{l^*(e)} \tag{7b}$$

where

$$l^*(e) = \frac{p}{q}(l(e) + c(i) - c(j)) + m_i - m_j. \tag{8}$$

By the definition of the phase function in (3), each $l^*(e)$ is an integer. Thus, the entries of an input matrix $A^*(D)$ are polynomials in D and D^{-1} with nonnegative integer coefficients. Note that from (8), l^* satisfies (4b). The lengths of edges are adjusted (by rescaling and adding/subtracting quantities that are "neutral" on each cycle) so that the input lengths are integral and are scaled correctly by the rate, p/q. (Note that the m_i are somewhat arbitrary; for example, setting each $m_i = 0$ is perfectly fine. However, the flexibility of choosing nontrivial m_i's turns out to be very useful in practice for shortening the code construction process. This will become apparent in Section IV).

As you would expect, not every adjacency matrix $A(D)$ of a VLG has an input matrix $A^*(D)$ for which the function l^* is an input length function. First, the matrix $A^*(D)$ need not be an adjacency matrix (i.e., it may have zero or negative exponents, $l^*(e) \leq 0$) and second, the exponents $l^*(e)$ need not satisfy (5). One necessary requirement for the existence of an input length function is that the capacity satisfies $C(\mathcal{S},l) \geq (p/q)\log N$. If p and q are relatively prime, then another necessary condition is that q divides the period $P(\mathcal{S},l)$. We show that these conditions are also sufficient by providing an algorithm for finding an equivalent VLG (i.e., a graph that describes the same constraint) with an input length function. In general, given that the capacity and period requirements are satisfied, some operations must be performed to alter the graph in order to obtain an input matrix that corresponds to an input length function. These alterations are guided by an approximate eigenvector. These procedures will always produce a VLG that is *N-codable at rate* $(p:q)$.

C. Approximate Eigenvectors

Let α be a positive real number. Let $A(D)$ be a square matrix with polynomial entries in D and D^{-1} with nonnegative, integer coefficients (e.g., the adjacency matrix, $A(D)$, for a VLG or an input matrix, $A^*(D)$.) An *α-approximate eigenvector* for $A(D)$ is a *positive* real vector $\boldsymbol{x}$ such that

$$A(\alpha^{-1})\boldsymbol{x} \geq \boldsymbol{x}.$$

We are most interested in α of the form $N^{p/q}$ or N. Let $\boldsymbol{y}$ be a positive integer vector and, with diagonal $V(D)$ as in (6), define

$$\boldsymbol{x} = V(N)\boldsymbol{y} \tag{9a}$$

or equivalently,

$$x_i = y_i N^{m_i + c(i)p/q}. \tag{9b}$$

Then $\boldsymbol{x}$ is an $N^{p/q}$-approximate eigenvector for $A(D)$

$$A(N^{-p/q})\boldsymbol{x} \geq \boldsymbol{x}, \tag{10a}$$

if and only if $\boldsymbol{y}$ is an N-approximate eigenvector for $A^*(D)$

$$A^*(N^{-1})\boldsymbol{y} \geq \boldsymbol{y}. \tag{10b}$$

Equivalently, for all $i \in \mathcal{S}$

$$\sum_j \sum_{e \in \mathcal{E}_{i,j}} N^{-pl(e)/q} x_j \geq x_i,$$

if and only if

$$\sum_j \sum_{e \in \mathcal{E}_{i,j}} N^{-l^*(e)} y_j \geq y_i.$$

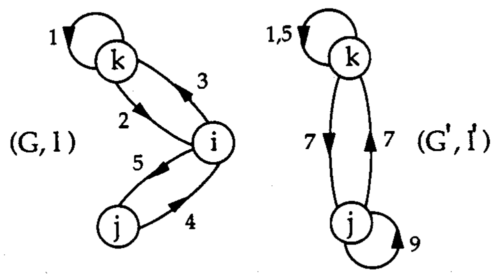

Fig. 5. Example of fusing state i.

Lemma 1: Let N be a positive integer and p,q a relatively prime pair of positive integers. If the capacity satisfies $C(\mathscr{G},l)\geq(p/q)\log N$ and q divides the period $P(\mathscr{G},l)$, then $(\mathscr{G},l)$ has a $N^{p/q}$-approximate eigenvector x such that for each state i, x_i is the product of a positive integer $y_i N^{m_i}$ and $N^{c(i)p/q}$, where $c(i)$ is the phase function described in (3). Moreover, each x_i can be expressed as in (9b), where N does not divide y_i ($y_i>0$ is a positive integer and $m_i\geq 0$ is a nonnegative integer).

Proof of Lemma 1: Let $V(D)$ be as in (6) with any choice of nonnegative $m_i\geq 0$ (e.g., $m_i=0$). Let $A^*(D)$ be as in (7). By linear algebra, the determinant $\det(I-A^*(D^{-1}))$ is equal to the determinant $\det(I-A^*(D^{-p/q}))$ and, therefore, its largest root is at least N. So, apply [7, Proposition 4] to $A^*(D)$. This works even though $A^*(D)$ may have entries with negative powers of D. This yields a positive integer solution y to (10b). Now, let x be defined as in (9) (with the same choice of m_i). Absorbing the powers of n that divide y_i into N^{m_i} yields the desired form. □

Remark 1: Let x be an $N^{p/q}$-approximate eigenvector in the form specified by the lemma; that is, each x_i is expressed as in (9b) with y_i not divisible by N. We call the vector $y=\{y_i\}$ the *(x,A)-induced vector* and $A^*(D)$ (as in (7)) the *(x,A)-induced input matrix.*

An iterative method (due to Franaszek [4, Appendix] and described in [2, Appendix]) can be used to find approximate eigenvectors for VLG's, as outlined in Section IV.

D. Fusing, Pruning, and State Splitting

A *fusion* of a VLG $(\mathscr{G},l)$ is a VLG $(\mathscr{G}',l')$ obtained from $(\mathscr{G},l)$ as follows. Let state $i\in\mathscr{S}$ have no self-loops (i.e., $|\mathscr{E}_{i,i}|=0$). Delete state i and all of its incident (incoming and outgoing) edges; for each pair of states $j,k\in\mathscr{S}$ $(j,k\neq i)$, and each pair of edges $(e,f)\in\mathscr{E}_{j,i}\times\mathscr{E}_{i,k}$, insert an edge g from state j to state k with length equal to $l'(g)=l(e)+l(f)$. We then say that *state i has been fused.* A labeled graph is fused in an analogous fashion; the label $w(g)=w(e)*w(f)$, where the asterisk is the concatenation operator. Fig. 5 shows an example of the fusing operation, the numbers on the edges represent the edge lengths.

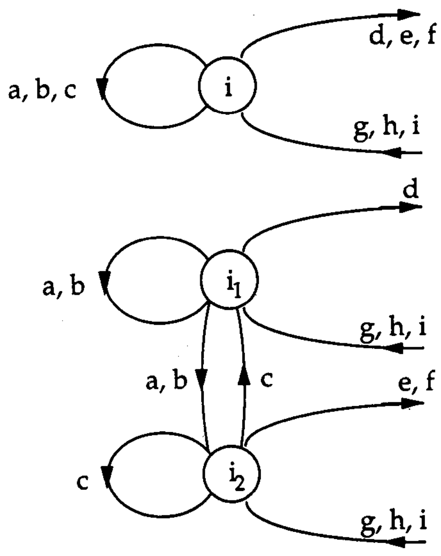

Fig. 6. Example of splitting state i.

A *pruning* of a VLG means the procedure of obtaining a subgraph of the VLG (i.e., removing edges and/or states).

Let x be a vector indexed by the states of a VLG $(\mathscr{G},l)$. Both operations, fusion and pruning, yield a new VLG whose states $\mathscr{S}'$ are a subset of the states of $\mathscr{G}$. The vector x' defined by

$$x_i'=x_i,\qquad \text{for all } i\in\mathscr{S}',$$

is called the *inherited vector of x.*

Suppose a VLG $(\mathscr{G}',l')$ is obtained from a VLG $(\mathscr{G},l)$ by a fusing operation. If x is an α-approximate eigenvector for $(\mathscr{G},l)$, then the inherited vector x' is an α-approximate eigenvector for $(\mathscr{G}',l')$. In addition, if the VLG $(\mathscr{G},l)$ is irreducible, so is the VLG $(\mathscr{G}',l')$. For the pruning operation, neither of these statements need hold. In both the fusing and pruning operations, however, the right-closing and right-resolving properties of a labeled graph are preserved.

By a *state splitting* of a VLG $(\mathscr{G},l)$ we mean a new VLG constructed as follows: Split a state i of $\mathscr{G}$ into two descendant states i_1 and i_2 by partitioning $\mathscr{E}_i$, the set of outgoing edges from i, into two *disjoint* sets and assigning one set $\mathscr{E}_{i_1}$ to i_1, and the other set $\mathscr{E}_{i_2}$ to i_2. Next, replicate incoming edges to i_1 and i_2. In Fig. 6, we have split state i according to the partition $\mathscr{E}_{i_1}=\{a,b,d\}$ and $\mathscr{E}_{i_2}=\{c,e,f\}$. Note that the self-loops at state i are both incoming as well as outgoing edges, and thus they are replicated.

Note that after splitting, the number of outgoing edges satisfies

$$\begin{aligned}|\mathscr{E}_i|=|\cup_j\mathscr{E}_{i,j}|&=|\cup_j\mathscr{E}_{i_1,j}|+|\cup_j\mathscr{E}_{i_2,j}|-|\mathscr{E}_{i,i}|\\&=|\mathscr{E}_{i_1}|+|\mathscr{E}_{i_2}|-|\mathscr{E}_{i,i}|\end{aligned}$$

and the number of incoming edges satisfies

$$|\cup_k\mathscr{E}_{k,i}|=|\cup_k\mathscr{E}_{k,i_1}|=|\cup_k\mathscr{E}_{k,i_2}|.$$

The new graph inherits a length function (and a labeling) in the obvious way. If the original labeling is right-closing, then the inherited labeling is also right-closing. However, if the original labeling is right-resolving, then the inherited labeling is right-closing, but it need not be right-resolving.

State splittings can be described in terms of the $n \times n$ adjacency matrix, $A(D)$, by splitting a row and copying the corresponding column (corresponding to splitting outgoing edges and replicating incoming edges), producing an $(n+1)\times(n+1)$ adjacency matrix $A'(D)$. To be precise, let $R_i(D)$ denote row i of $A(D)$. Any splitting of state i determines a decomposition:

$$R_i(D) = R_{i_1}(D) + R_{i_2}(D),$$

where $R_{i_1}(D), R_{i_2}(D)$ are rows with polynomials in D with nonnegative integer coefficients. Let

$$B(D) = \begin{pmatrix} R_1(D) \\ R_2(D) \\ \vdots \\ R_{i-1}(D) \\ R_{i_1}(D) \\ R_{i_2}(D) \\ R_{i+1}(D) \\ \vdots \\ R_n(D) \end{pmatrix}$$

and let $C_j(D)$ denote column j of $B(D)$. Then

$$A'(D) = (C_1(D) \quad C_2(D) \quad \cdots \quad C_{i-1}(D) \quad C_i(D) \quad C_i(D) \quad C_{i+1}(D) \quad \cdots \quad C_n(D))$$

is the adjacency matrix of the VLG obtained by the splitting.

E. Tight Approximate Eigenvectors and N-codable VLG's

The procedures of fusing, pruning, and state-splitting can be applied to produce encodable VLG's, as illustrated in Section IV. In this section and Section III-F, we prove the basic coding result:

A VLG $(\mathscr{S}, l)$ can be "transformed" into a VLG $(\bar{\mathscr{S}}, \bar{l})$ that is N-codable at rate $(p:q)$ if and only if the capacity satisfies $C(\mathscr{S}, l) \geq (p/q)\log N$.

The proof involves the notion of a tight approximate eigenvector, which we define next. The proof uses an iterative state-splitting procedure in which, after each iteration, we need to ensure that the resulting approximate eigenvector is tight. (In practice, however, it is our experience that it is usually not necessary to have a tight eigenvector before splitting in order to carry out a successful code construction).

Let $\boldsymbol{x}$ be an α-approximate eigenvector for a VLG $(\mathscr{S}, l)$. We say that $\boldsymbol{x}$ is *tight* if for all $i, j \in \mathscr{S}$ and all edges $e \in \mathscr{E}_{i,j}$

$$\alpha^{-l(e)} x_j < x_i.$$

Note that if $\boldsymbol{x}$ is tight, any pruning operation must retain at least 2 outgoing edges from each state in order to ensure that the inherited vector satisfies the α-approximate eigenvector inequality.

Lemma 2: Let $(\mathscr{S}, l)$ be an irreducible VLG. Let $\boldsymbol{x}$ be an α-approximate eigenvector, with $\alpha > 1$. Then there is an irreducible VLG $(\mathscr{S}', l')$ obtained from $(\mathscr{S}, l)$ by a sequence of fusing/pruning operations such that the inherited vector $\boldsymbol{x}'$ is a tight α-approximate eigenvector.

Proof of Lemma 2: Suppose $\boldsymbol{x}$ itself is not tight. Then, there are states i, j and an edge $e \in \mathscr{E}_{i,j}$ such that

$$\alpha^{-l(e)} x_j \geq x_i.$$

Since $\alpha > 1$, the edge e is not a self-loop. If $|\mathscr{E}_i| = 1$, then state i has no self-loops and we can fuse state i to produce a VLG $(\mathscr{S}', l')$. As mentioned in Section III-D, $(\mathscr{S}', l')$ is an irreducible VLG and the inherited vector $\boldsymbol{x}'$ is still an α-approximate eigenvector.

If, on the other hand, $|\mathscr{E}_i| \geq 2$, then let $(\mathscr{S}', l')$ be the subgraph of $(\mathscr{S}, l)$ obtained by pruning all outgoing edges from i other than e. The inherited vector $\boldsymbol{x}'$ is still an α-approximate eigenvector, but $(\mathscr{S}', l')$ may fail to be irreducible. However, any graph must have a *sink component*, an irreducible subgraph such that all edges that begin in the subgraph must end in the subgraph. Replacing $(\mathscr{S}', l')$ with a sink component of $(\mathscr{S}', l')$ yields an irreducible subgraph whose inherited vector $\boldsymbol{x}'$ is still an α-approximate eigenvector.

In both cases, fusing or pruning, the operation produces an irreducible VLG with a smaller number of edges than $(\mathscr{S}, l)$ and whose inherited vector is still an α-approximate eigenvector. Since the graph is finite, this process must terminate with a tight α-approximate eigenvector. □

Lemma 3: Let $(\mathscr{S}, l)$ be an irreducible VLG with an $N^{p/q}$-approximate eigenvector $\boldsymbol{x}$ and induced vector (as in Remark 1) $\boldsymbol{y} = \mathbf{1}$, the all 1's vector. Then, there is a VLG, obtained from $(\mathscr{S}, l)$ by a sequence of fusing/pruning operations, which is N-codable at rate $(p:q)$.

Proof of Lemma 3: By Lemma 2, it may be assumed that $\boldsymbol{x}$ is tight. Thus, for all edges $e \in \mathscr{E}_{i,j}$,

$$N^{-(p/q)l(e)} x_j < x_i,$$

which is equivalent to (see the definition (8) of $l^*(e)$)

$$N^{-l^*(e)} y_j < y_i.$$

But since $\boldsymbol{y} = \mathbf{1}$ (each $y_i = 1$), each length $l^*(e)$ is positive. Combining this with the form (8) of $l^*(e)$, it is immediate that l^* satisfies (4a) and (4b).

The hypothesis $\boldsymbol{y} = \mathbf{1}$ yields, for each state $i \in \mathscr{S}$,

$$\sum_{e \in \mathscr{E}_i} N^{-l^*(e)} \geq 1.$$

This, together with the fact that each $l^*(e)$ is a positive integer, implies that for each state $i \in \mathscr{S}$ there is a subset of outgoing edges $\bar{\mathscr{E}}_i \subset \mathscr{E}_i$ such that (5) is satisfied

$$\sum_{e \in \bar{\mathscr{E}}_i} N^{-l^*(e)} = 1.$$

Now, replace $(\mathscr{G}, l)$ by the subgraph obtained by pruning all other edges, i.e., all but $\cup_{i \in \mathscr{S}} \bar{\mathscr{E}}_i$. This subgraph is N-codable at rate $(p:q)$. □

F. Basic Result: Obtaining N-Codable VLG's

In this section, we prove the basic coding theorem for VLG's. We first define the concept of a q-fold trellis, to which we refer in the proof.

A *q-fold trellis* is a graph $\mathscr{G}_q = (\mathscr{S} \times \{0, 1, \cdots, q-1\}, \mathscr{E}')$ obtained from the graph $\mathscr{G} = (\mathscr{S}, \mathscr{E})$. If edge e goes from state i to state j in the original graph, then for each state (i, k), $0 \le k < q$, there is a copy of the edge e going to $(j, k + l(e)$ modulo $q)$. Note that the capacity of the q-fold trellis is the same as the original graph, and its period is divisible by q. If the original graph has a labeling, then the q-fold trellis naturally inherits a labeling that describes the same constraint.

Theorem 2: Let $(\mathscr{G}, l)$ be a VLG, N a positive integer, and p, q be a relatively prime pair of positive integers. Suppose also that $C(\mathscr{G}, l) \ge (p/q)\log N$, and that q divides the period, $P(\mathscr{G}, l)$. Let x be a $(N^{p/q})$-approximate eigenvector with induced vector y (as in Remark 1). Then there exists a VLG, $(\bar{\mathscr{G}}, \bar{l})$, obtained from $(\mathscr{G}, l)$ by a sequence of fusing, pruning, and at most $\Sigma_i(y_i - 1)$ state splitting operations such that:

$$(\bar{\mathscr{G}}, \bar{l}) \text{ is } N\text{-codable at rate } (p:q)$$

and

$$\bar{\mathscr{G}} \text{ has at most } \sum_i y_i \text{ states.}$$

Discussion: The proof of the theorem is obtained by iteratively applying state splitting followed by fusing/pruning operations until a graph which is N-codable at rate $(p:q)$ is produced. The idea is that, in each step of the splitting process, a certain state i is identified and split into descendant states i_1, i_2 such that the ith component y_i of the induced vector is the sum of two strictly smaller positive integers, $y_i = y_{i_1} N^{m_{i_1}} + y_{i_2} N^{m_{i_2}}$. Thus, the sizes of the components of the induced vector monotonically decrease until each component satisfies $y_i = 1$. Note that it is desirable to find large factors of N in the split components since these factors are "absorbed" into m_{i_1}, m_{i_2} of the induced vector (Remark 1). For example, if $N = 2$, $y_i = 5$, and y_i is split into $5 = 4 + 1$, then $y_{i_1} = y_{i_2} = 1$, $m_{i_1} = 2$ and $m_{i_2} = 0$; in this case, no further splitting is required for either descendant state i_1 or i_2. If, on the other hand, the splitting is $5 = 3 + 2$, then $y_{i_1} = 3 > 1$, $y_{i_2} = 1$, $m_{i_1} = 0$ and $m_{i_2} = 1$, and state i_1 requires further splitting.

Once y has been reduced to the all 1's vector, we apply Lemma 3 to produce a VLG that is N-codable at rate $(p:q)$. The choice of splitting is determined by Proposition 1, which we state after a brief discussion of code construction. The proof of Proposition 1 is obtained by reduction to [7].

One uses Theorem 2 to construct codes as follows: Let a constrained system be presented by a right-closing labeling of a VLG $(\mathscr{G}, l)$ with capacity $C(\mathscr{G}, l) \ge (p/q)\log N$ (a necessary condition for a rate $(p:q)$ code). If $C(\mathscr{G}, l) = (p/q)\log N$ then the period assumption, q divides $P(\mathscr{G}, l)$, can be deduced from the *Perron–Frobenius Theory* (See [16, 17]). Otherwise, if needed, the period assumption can be met by enlarging the VLG into a q-fold trellis, to which we can then apply the theorem. Since $(\bar{\mathscr{G}}, \bar{l})$ is N-codable at rate $(p:q)$, the free N-ary source can be encoded into the constraint. Since splitting, fusing, and pruning preserve the right-closing property, the VLG $(\bar{\mathscr{G}}, \bar{l})$ produced by Theorem 2 inherits a right-closing labeling from the original right-closing labeling on $(\mathscr{G}, l)$, and decoding can be performed in a state-dependent manner, with delay.

In many cases, the method described here shortens the code construction procedures described in previous papers. For example, in the case $p = q = 1$, a VLG is N-codable at rate $(p:q)$ if it has an approximate eigenvector, all of whose components are simply powers of N. As previously observed, in such a case one can construct a rate $(1:1)$ code without any splitting at all, in contrast to the method described in [7]. In other cases, splitting is required by the new method, but typically not as many splittings are needed. For instance, splitting a state so that the ith component y_i of the induced vector is split into two summands, at least one of which is a power of N, shortens the splitting process. While the same codes may be found by using the previous state splitting methods ([2], [7]), the computational procedure produced here is often shorter and produces reasonably simple codes.

Proposition 1: Let $(\mathscr{G}, l)$, N, p, q and x, y be as in Theorem 2. Suppose that x is tight. Then either $(\mathscr{G}, l)$ has a subgraph with an induced vector of all ones (i.e., $y = \mathbf{1}$ for the subgraph) or for some state i, there is a decomposition of the ith row of the induced matrix, $A^*(D)$, into two rows, $R^*_{i_1}(D), R^*_{i_2}(D)$, (polynomials in D and D^{-1} with nonnegative, integer coefficients)

$$R^*_i(D) = R^*_{i_1}(D) + R^*_{i_2}(D)$$

and positive integers u, v such that

$$R^*_{i_1}(N^{-1})y \ge u, \qquad R^*_{i_2}(N^{-1})y \ge v \tag{11a}$$

and

$$u + v = y_i. \tag{11b}$$

The proof of Proposition 1 is given at the end of the Appendix.

Proof of Theorem 2: By Lemma 2, we may assume that x is tight and, by Lemma 3, that, for all subgraphs, the induced vector $y \ne \mathbf{1}$. Now apply Proposition 1. The decomposition of $R^*_i(D)$ naturally determines a partition of the outgoing edges $\mathscr{E}_i$ of state i in $(\mathscr{G}, l)$ for a state splitting operation. The vector x' defined by:

$$x'_j = \begin{cases} uN^{m_i + c(i)p/q}, & j = i_1; \\ vN^{m_i + c(i)p/q}, & j = i_2; \\ x_j, & \text{otherwise}; \end{cases}$$

is easily checked to be an $N^{p/q}$-approximate eigenvector for $(\mathscr{G}', l')$, the VLG produced by this splitting. Write $u = y_{i_1} N^{m_{i_1}}$ and $v = y_{i_2} N^{m_{i_2}}$, where N does not divide either y_{i_1} or y_{i_2}. Then, the vector y' induced by x' is given by:

$$y'_j = \begin{cases} y_{i_1}, & j = i_1; \\ y_{i_2}, & j = i_2; \\ y_j, & \text{otherwise}. \end{cases}$$

Since $\max\{u, v\} < y_i$, the process which iteratively splits $(\mathscr{G}, l)$ into $(\mathscr{G}', l')$ must terminate (i.e., at some point, for some subgraph, the induced vector, $y = 1$). At this point, Lemma 3 can be invoked to complete the procedure for finding the N-codable VLG. Clearly, the number of iterations is at most

$$\sum_i (y_i - 1). \qquad \Box$$

G. Decoding

We briefly discuss the decoding problem. Since the labelings of interest are right-closing, the role of input/output can be reversed to convert the finite-state encoder into a finite-state decoder with delay. However, it is often desirable for the decoder to be a sliding-block mapping, i.e., the decoded p-block should depend on only a bounded amount of memory/anticipation of the input to the decoder. This property guarantees limited error propagation when the code is used on a noisy channel.

The problem of constructing sliding-block decoders has been studied in detail in [18], and is, in general, very difficult. However, it is quite tractable in the case of finite-memory constraints, which we now discuss.

A labeling has *finite memory* if the mapping from bi-infinite paths to bi-infinite sequences, generated by the labeling function $w: \mathscr{E} \to \mathscr{W}$, is one-to-one. A finite-memory constraint is one that can be presented by a finite-memory labeling. It is easy to see that the finite memory property is preserved by fusing, pruning, and state splitting.

Now, if a VLG is N-codable at rate $(p:q)$ and has a labeling that has finite memory, then the decoder can be made sliding-block—roughly because the finite memory property implies that for sufficiently long words w, all the paths that generate w must agree at some time. When designing a rate $(p:q)$ code, the construction algorithm based upon Theorem 2 may require the replacement of a VLG by its q-fold trellis. Even if the VLG has a finite-memory labeling, the corresponding labeling of the q-fold trellis need not. However, if we add to the edge labels in the q-fold trellis an indicator of the accumulated number of symbols modulo q, then the labeling will have finite memory. Strictly speaking, this changes the constraint, but in practice the decoding of a rate $(p:q)$ code requires the availability of this phase information. Thus, for any finite-memory constraint, the encoders constructed in this paper have sliding-block decoders. While RLL constraints have finite memory, the ARC constraints, in general, do not. However, for certain ARC constraints, there are constructions of decoders that are effectively sliding-block, provided that certain physically measurable quantities are available. This is illustrated by an example in Section IV-C.

IV. Examples of ARC and Variable-Length State Splitting

A. Algorithm for Finding Approximate Eigenvectors

We first recall an algorithm due to Franaszek to find approximate eigenvectors [2], [4]. Given a VLG $(\mathscr{G}, l)$ with $C(\mathscr{G}, l) \geq (p/q)\log N$, and q dividing $P(\mathscr{G}, l)$, the procedure finds an $N^{p/q}$-approximate eigenvector. With each $m_i = 0$ (in (6) of Section III), let input matrix $A^*(D)$ be as in (7) (of Section III). Then, with some initial choice of a positive integer vector $y^{(0)}$, iterate

$$y_i^{(m+1)} = \min\left(y_i^{(m)}, \left\lfloor \left(A^*(N^{-1})y^{(m)}\right)_i \right\rfloor\right),$$

where $\lfloor \cdot \rfloor$ denotes the greatest integer function.

The sequence $y^{(m)}$ consists of nonincreasing, nonnegative integer vectors. Therefore it must eventually stabilize to some vector y. For this limiting vector,

$$A^*(N^{-1})y \geq y.$$

Let x be the nonnegative vector

$$x_i = y_i N^{c(i)p/q}.$$

Then, x is an $N^{p/q}$-approximate eigenvector provided that all of its components are strictly positive. If some components are 0, then simply prune to the subgraph determined by the states whose components are strictly positive. Unfortunately, x may be the all 0's vector, in which case the result is the empty graph. Fortunately, this will not happen if the initial choice $y^{(0)}$ is sufficiently large. Note that often one can get a smaller y, namely the induced vector y determined by x as in Remark 1 (i.e., separate out from y_i the powers of N that divide y_i). Also note that the procedure sometimes finds an approximate eigenvector for a proper subgraph of the original VLG, since some of the components are zero. This is certainly good enough for coding purposes.

B. A VLG Approach to "(2,7)"

As an example of the variable-length state-splitting technique, consider the $(d,k) = (2,7)$ constraint, with $N = 2$. The algorithm proceeds as follows. First, for the variable-length graph in Fig. 7, the adjacency matrix is

$$A_1(D) = D^3 + D^4 + D^5 + D^6 + D^7 + D^8,$$

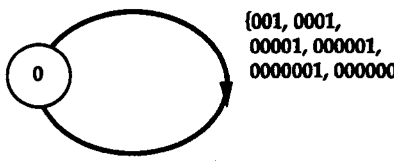

Fig. 7. Variable-length representation of $(d,k)=(2,7)$ constraint.

and the largest root of

$$\det(I-A_1(D^{-1}))$$
$$=1-(D^{-3}+D^{-4}+D^{-5}+D^{-6}+D^{-7}+D^{-8})$$

is $\lambda=1.431346\cdots$, so the capacity $C=\log_2(\lambda)=0.517372\cdots$. This means that it is possible to encode binary data into the $(d,k)=(2,7)$ constraint at any rate less than $0.517372\cdots$. An obvious choice is to encode at rate $(p:q)=(1:2)$. Note that the graph is aperiodic; that is, $P(\mathscr{G},l)=1$.

Before the state splitting algorithm can be applied, the graph must be made periodic with period $P(\mathscr{G},l)=q=2$. The 2-fold trellis is shown in Fig. 8. For this graph, the adjacency matrix is

$$A_2(D)=\begin{pmatrix} D^4+D^6+D^8 & D^3+D^5+D^7 \\ D^3+D^5+D^7 & D^4+D^6+D^8 \end{pmatrix}.$$

This VLG has period 2; we choose the phase function $c(0)=0$, $c(1)=1$. The reader may check that applying the approximate eigenvector algorithm with $y^{(0)}=\binom{3}{3}$ yields

$$x=\begin{pmatrix} 3 \\ 2^{3/2} \end{pmatrix}=\begin{pmatrix} 3\cdot 2^0 \\ 1\cdot 2^{3/2} \end{pmatrix}=\begin{pmatrix} 3\cdot 2^{0+0\cdot(1/2)} \\ 1\cdot 2^{1+1\cdot(1/2)} \end{pmatrix},$$

a $2^{1/2}$-approximate eigenvector. Then x and $A_2(D)$ determine the induced vector

$$y=\begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

and induced input matrix

$$\begin{aligned} A_2^*(D) &= V(D)A(D^{1/2})V(D^{-1}) \\ &= \begin{pmatrix} D^0 & 0 \\ 0 & D^{3/2} \end{pmatrix} \\ &\quad \cdot \begin{pmatrix} D^2+D^3+D^4 & D^{3/2}+D^{5/2}+D^{7/2} \\ D^{3/2}+D^{5/2}+D^{7/2} & D^2+D^3+D^4 \end{pmatrix} \\ &\quad \cdot \begin{pmatrix} D^0 & 0 \\ 0 & D^{-3/2} \end{pmatrix} \\ &= \begin{pmatrix} D^2+D^3+D^4 & D^0+D^1+D^2 \\ D^3+D^4+D^5 & D^2+D^3+D^4 \end{pmatrix} \end{aligned}$$

satisfying

$$A_2^*(2^{-1})y\geq y.$$

Split the graph according to the following decomposition

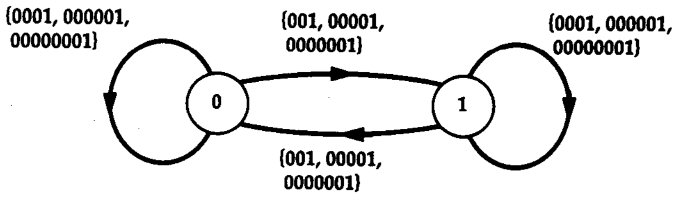

Fig. 8. Period-2, $(d,k)=(2,7)$ constraint.

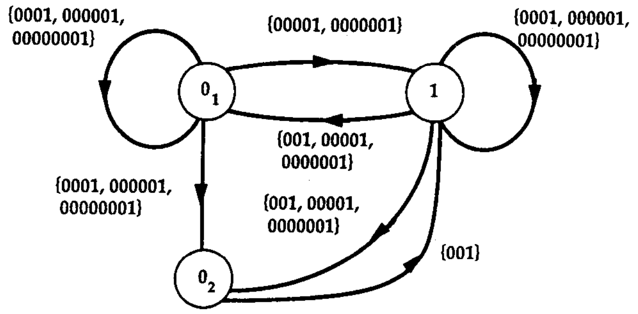

Fig. 9. Split graph, $(d,k)=(2,7)$ constraint.

of the first row of $A_2^*(D)$ (see Proposition 1)

$$(D^2+D^3+D^4, D^0+D^1+D^2)$$
$$=(D^2+D^3+D^4, D^1+D^2)+(0,D^0).$$

Letting $u=2$, $v=1$, this decomposition satisfies (11) of Section III:

$$(2^{-2}+2^{-3}+2^{-4}, 2^{-1}+2^{-2})\begin{pmatrix} 3 \\ 1 \end{pmatrix}\geq 2$$

and

$$(0,1)\begin{pmatrix} 3 \\ 1 \end{pmatrix}\geq 1.$$

Note that the term D^0 from the first row of $A_2^*(D)$ is split off. This corresponds to splitting off the term D^3 from the first row of $A_2(D)$ and therefore determines a partition

$$\mathscr{E}_0=\mathscr{E}_{0_1}\cup\mathscr{E}_{0_2},$$

where $\mathscr{E}_{0_2}$ consists of the single edge of length 3 from state 0 to state 1. This partition gives the new labeled VLG, obtained by splitting, as shown in Fig. 9. The new adjacency matrix is

$$A_3(D)$$
$$=\begin{pmatrix} D^4+D^6+D^8 & D^4+D^6+D^8 & D^5+D^7 \\ 0 & 0 & D^3 \\ D^3+D^5+D^7 & D^3+D^5+D^7 & D^4+D^6+D^8 \end{pmatrix}$$

with approximate eigenvector

$$x=\begin{pmatrix} 2 \\ 1 \\ 2^{3/2} \end{pmatrix}=\begin{pmatrix} 1\cdot 2^1 \\ 1\cdot 2^0 \\ 1\cdot 2^{3/2} \end{pmatrix}.$$

Note that the first component, satisfying $x_0=3$, has been split into summands $x_{0_1}=2$ and $x_{0_2}=1$, and that both descendant states $0_1, 0_2$ have the same phase as their

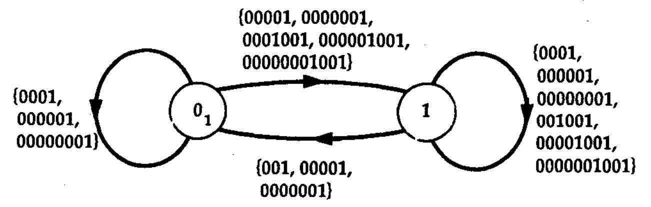

Fig. 10. Fused graph, $(d,k)=(2,7)$ constraint.

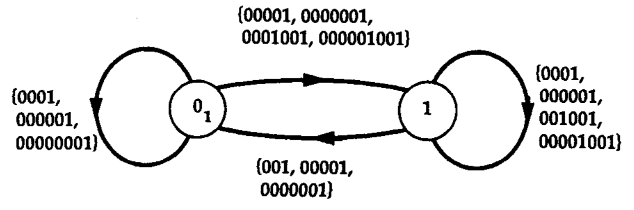

Fig. 11. Pruned subgraph, $(d,k)=(2,7)$ constraint.

parent state 0. The corresponding induced vector is given by

$$y=\begin{pmatrix}1\\1\\1\end{pmatrix}$$

and the corresponding induced input matrix is

$$A_3^*(D)=\begin{pmatrix}D^2+D^3+D^4 & D^3+D^4+D^5 & D^2+D^3\\ 0 & 0 & D^0\\ D^2+D^3+D^4 & D^3+D^4+D^5 & D^2+D^3+D^4\end{pmatrix}.$$

According to Lemma 3, a VLG which is 2-codable at rate (1:2) can be obtained by applying the fusion and pruning operations to the new VLG. Fusing state 0_2 gives the VLG shown in Fig. 10. Its adjacency matrix is

$$A_4(D)=\begin{pmatrix}D^4+D^6+D^8 & D^5+2D^7+D^9+D^{11}\\ D^3+D^5+D^7 & D^4+2D^6+2D^8+D^{10}\end{pmatrix}.$$

The inherited $2^{1/2}$-approximate eigenvector is

$$x=\begin{pmatrix}2\\2^{3/2}\end{pmatrix}=\begin{pmatrix}1\cdot 2^1\\1\cdot 2^{3/2}\end{pmatrix}$$

and the corresponding induced vector is

$$y=\begin{pmatrix}1\\1\end{pmatrix}.$$

The induced input matrix is

$$A_4^*(D)=\begin{pmatrix}D^2+D^3+D^4 & D^2+2D^3+D^4+D^5\\ D^2+D^3+D^4 & D^2+2D^3+2D^4+D^5\end{pmatrix}.$$

Finally, passing to the subgraph, we obtain the VLG shown in Fig. 11, with adjacency matrix

$$A_5(D)=\begin{pmatrix}D^4+D^6+D^8 & D^5+2D^7+D^9\\ D^3+D^5+D^7 & D^4+2D^6+D^8\end{pmatrix}$$

and induced input matrix

$$A_5^*(D)=\begin{pmatrix}D^2+D^3+D^4 & D^2+2D^3+D^4\\ D^2+D^3+D^4 & D^2+2D^3+D^4\end{pmatrix}.$$

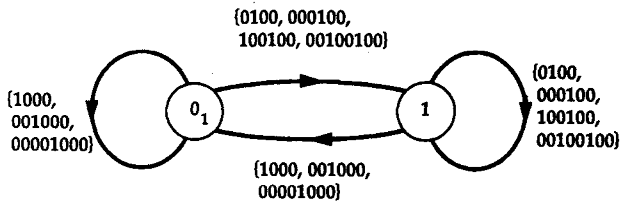

Fig. 12. Shifted graph, $(d,k)=(2,7)$ constraint.

This VLG is 2-codable at rate (1:2) because the row sums of $A_5^*(2^{-1})$ are both 1, satisfying (5). An encoder can be made by assigning a complete prefix-free list of (input) binary words of lengths 2,3,4,2,3,3,4 (i.e., the exponents (with multiplicity) that appear in the rows of $A_5^*(D)$) to the outgoing edges at each state. Namely, at state 0_1 assign these input words to the outgoing edges of length 4,6,8,5,7,7,9, (the exponents that appear in row 1 of $A_5(D)$) and at state 1 assign these input words to the outgoing edges of length 3,5,7,4,6,6,8 (the exponents that appear in row 2 of $A_5(D)$). Note that this encoder has rate (1:2) on each cycle.

One can always transform such an encoder to one which is of rate (1:2) on each edge by a combination of shifting labels and splitting incoming edges. In fact, we shift the labels of the encoder graph corresponding to $A_5(D)$ in the following way: three (respectively, two) symbols of the strings on the outgoing edges of state 0_1 (respectively, 1) are shifted off of the head of each string and three (respectively, two) symbols of the strings on the incoming edges of state 0_1 (respectively, 1) are shifted on to the tail of each string. This yields a new labeled graph, shown in Fig. 12, whose adjacency matrix is conjugate to $A_5(D)$:

$$\begin{aligned}A_6(D)&=\begin{pmatrix}D^{-3} & 0\\ 0 & D^{-2}\end{pmatrix}\\ &\cdot\begin{pmatrix}D^4+D^6+D^8 & D^5+2D^7+D^9\\ D^3+D^5+D^7 & D^4+2D^6+D^8\end{pmatrix}\begin{pmatrix}D^3 & 0\\ 0 & D^2\end{pmatrix}\\ &=\begin{pmatrix}D^4+D^6+D^8 & D^4+2D^6+D^8\\ D^4+D^6+D^8 & D^4+2D^6+D^8\end{pmatrix}.\end{aligned}$$

This graph has the nice property that the set of codewords is the same for each state, {0100, 1000, 000100, 001000, 100100, 00100100, 00001000}. Since

$$(1/2)^2+(1/2)^2+(1/2)^3+(1/2)^3+(1/2)^3+(1/2)^4+(1/2)^4=1,$$

we can define an encoder simply by choosing a complete, prefix-free list of binary words having lengths {2,2,3, 3,3,4,4}, and choosing a 1-to-1 correspondence with the codeword list. The resulting code is the Franaszek (2,7) code that is in wide use in commercial storage devices [3]. It should be noted that the last step of the construction, which yields this extremely simple encoder, is somewhat *ad hoc*.

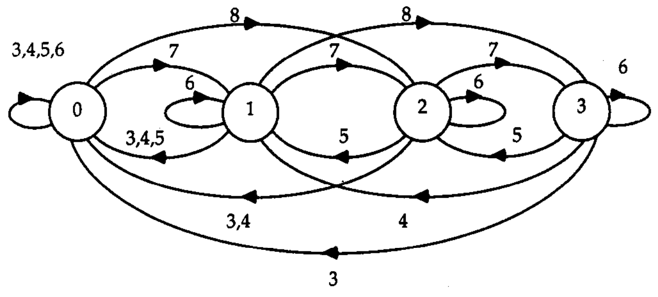

Fig. 13. $(d,k,a,b)=(2,7,6,3)$ ARC.

C. A $(d,k,a,b)=(2,7,6,3)$ ARC Code

Because a runlength of 8 labels a self-loop in the graph in Fig. 11, the $(d,k)=(2,7)$ encoder designed in the previous example can produce the output sequence 00001000, 00001000, 00001000, $\cdots$ that has an average runlength of $k+1=8$. It is, however, possible to improve the worst case average runlength to 6 at rate $(p:q)=(1:2)$. The $(d,k,a,b)=(2,7,6,3)$ ARC constraint is described by Fig. 13, where the numbers on the edges represent runlengths, and by the adjacency matrix

$$A(D)=\begin{pmatrix} D^3+D^4+D^5+D^6 & D^7 & D^8 & 0 \\ D^3+D^4+D^5 & D^6 & D^7 & D^8 \\ D^3+D^4 & D^5 & D^6 & D^7 \\ D^3 & D^4 & D^5 & D^6 \end{pmatrix}.$$

The capacity for this constraint is $0.515659\cdots$ ($\lambda=1.429647\cdots$).

This VLG has period 1. A $(d,k,a,b)=(2,7,6,3)$ ARC encoder can be produced by replacing this graph by its 2-fold trellis and splitting the resulting period-2 graph. Unfortunately, this doubles the number of states. However, an easier approach follows by considering a related constraint called the modulo average runlength constraint, or *MOD-ARC*. This constraint, described on the same set of states as the ARC constraint, defines a subset of the ARC sequences.

The MOD-ARC is conveniently described in terms of the graph with vertices labeled with the numbers $0,1,\cdots,b$. From each vertex $0\le i\le b$, draw an edge to vertex $0\le j\le b$, labeled with a runlength of length l, if

$$d+1\le l\le k+1$$
$$i+l-a\le b$$

and

$$j=i+l-a \qquad (\text{modulo } b+1).$$

To see that the MOD-ARC is contained in the ARC, we argue as follows. Let $T_1\cdots T_n$ be the sequence of runlengths of a MOD-ARC sequence. Let $S_1\cdots S_n$ be the state sequence in the MOD-ARC graph of a path that generates this sequence. Inductively, define the sequence S_n^* by

$$S_0^*=0$$
$$S_n^*=\max\{0,S_{n-1}^*+T_n-a\}.$$

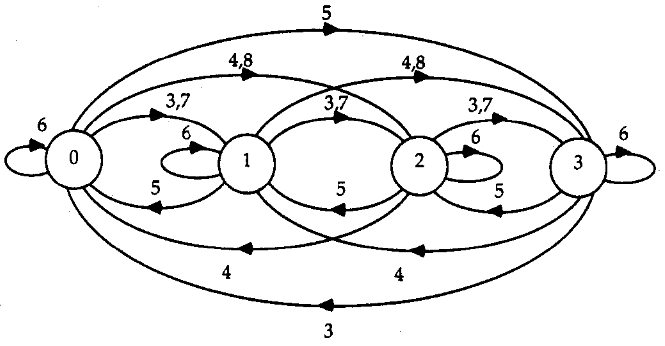

Fig. 14. $(d,k,a,b)=(2,7,6,3)$ MOD-ARC.

It suffices to show that $S_n^*\le S_n$ since this implies $S_n^*\le b$, and thus the runlength sequence can be generated by the ARC graph. This is proved inductively as follows.

If $S_n^*=0$, then clearly $S_n^*\le S_n$. Otherwise,

$$\begin{aligned} S_n^* &= S_{n-1}^*+T_n-a \\ &\le S_{n-1}+T_n-a \\ &\le S_n. \end{aligned}$$

The MOD-ARC $(d,k,a,b)=(2,7,6,3)$ is shown in Fig. 14 and has adjacency matrix

$$A_1(D)=\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ 0 & D^6 & D^3+D^7 & D^4+D^8 & D^5 \\ 1 & D^5 & D^6 & D^3+D^7 & D^4+D^8 \\ 2 & D^4 & D^5 & D^6 & D^3+D^7 \\ 3 & D^3 & D^4 & D^5 & D^6 \end{array}$$

Interestingly, the capacity of this graph is exactly $1/2$ (i.e., $\lambda=\sqrt{2}$). The period of the graph is 2. Thus, a rate (1:2) code can be found directly by splitting this graph. (We also mention the interesting fact that the sequences generated by the (1,7) codes developed independently by Jacoby and Adler–Moussouris–Hassner are described by the $(d,k,a,b)=(1,7,6,2)$ MOD-ARC, a graph with capacity 2/3 and period 3. See [11].)

Take the phase function $c(0)=c(2)=0$, $c(1)=c(3)=1$. Then

$$x=\begin{pmatrix} 13 \\ 9\lambda \\ 12 \\ 8\lambda \end{pmatrix}=\begin{pmatrix} 13\cdot 2^0 \\ 9\cdot 2^{1/2} \\ 3\cdot 2^2 \\ 1\cdot 2^{7/2} \end{pmatrix}$$

is an eigenvector of $A_1(2^{-1/2})$; that is, $A_1(2^{-1/2})x=x$. Since the capacity is exactly $1/2$, a $2^{1/2}$-approximate eigenvector is actually an eigenvector. Now, $A_1(D)$ and x determine the induced vector

$$y=\begin{pmatrix} 13 \\ 9 \\ 3 \\ 1 \end{pmatrix}$$

and the induced input matrix

$$A_1^*(D)=\begin{array}{c|cccc} & 0 & 1 & 2 & 3\\ \hline 0 & D^3 & D^1+D^3 & D^0+D^2 & D^{-1}\\ 1 & D^3 & D^3 & D^0+D^2 & D^{-1}+D^1\\ 2 & D^4 & D^4 & D^3 & D^0+D^2\\ 3 & D^5 & D^5 & D^4 & D^3 \end{array}$$

satisfying $A_1^*(2^{-1})y\geq y$. We give the results of two successive rounds of state splitting and the resulting adjacency matrices and eigenvectors, from which the reader can readily determine the explicit state splittings.

First, the graph is split into 8 states

$$A_2(D)=\begin{array}{c|cccccccc} & 0_1 & 0_2 & 1_1 & 1_2 & 1_3 & 2_1 & 2_2 & 3\\ \hline 0_1 & D^6 & D^6 & D^3+D^7 & D^3+D^7 & D^3+D^7 & D^8 & D^8 & 0\\ 0_2 & 0 & 0 & 0 & 0 & 0 & D^4 & D^4 & D^5\\ 1_1 & D^5 & D^5 & D^6 & D^6 & D^6 & D^7 & D^7 & D^8\\ 1_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^4\\ 1_3 & 0 & 0 & 0 & 0 & 0 & D^3 & D^3 & 0\\ 2_1 & D^4 & D^4 & D^5 & D^5 & D^5 & D^6 & D^6 & D^7\\ 2_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^3\\ 3 & D^3 & D^3 & D^4 & D^4 & D^4 & D^5 & D^5 & D^6 \end{array}$$

with the resulting eigenvector

$$x=\begin{pmatrix}8\\5\\4\lambda\\2\lambda\\3\lambda\\8\\4\\8\lambda\end{pmatrix}=\begin{pmatrix}1\cdot2^3\\5\cdot2^0\\1\cdot2^{5/2}\\1\cdot2^{3/2}\\3\cdot2^{1/2}\\1\cdot2^3\\1\cdot2^2\\1\cdot2^{7/2}\end{pmatrix}.$$

This new graph is further split into 10 states,

$$A_3(D)=\begin{array}{c|cccccccccc} & 0_1 & 0_{2,1} & 0_{2,2} & 1_1 & 1_2 & 1_{3,1} & 1_{3,2} & 2_1 & 2_2 & 3\\ \hline 0_1 & D^6 & D^6 & D^6 & D^3+D^7 & D^3+D^7 & D^3+D^7 & D^3+D^7 & D^8 & D^8 & 0\\ 0_{2,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^4 & 0 & D^5\\ 0_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^4 & 0\\ 1_1 & D^5 & D^5 & D^5 & D^6 & D^6 & D^6 & D^6 & D^7 & D^7 & D^8\\ 1_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^4\\ 1_{3,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^3 & 0\\ 1_{3,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^3 & 0 & 0\\ 2_1 & D^4 & D^4 & D^4 & D^5 & D^5 & D^5 & D^5 & D^6 & D^6 & D^7\\ 2_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^3\\ 3 & D^3 & D^3 & D^3 & D^4 & D^4 & D^4 & D^4 & D^5 & D^5 & D^6 \end{array}$$

with the resulting eigenvector

$$x=\begin{pmatrix}8\\4\\1\\4\lambda\\2\lambda\\\lambda\\2\lambda\\8\\4\\8\lambda\end{pmatrix}=\begin{pmatrix}1\cdot2^3\\1\cdot2^2\\1\cdot2^0\\1\cdot2^{5/2}\\1\cdot2^{3/2}\\1\cdot2^{1/2}\\1\cdot2^{3/2}\\1\cdot2^3\\1\cdot2^2\\1\cdot2^{7/2}\end{pmatrix}.$$

This eigenvector determines the induced vector y that consists entirely of 1's. So, using Lemma 3, fusion and pruning operations produce a graph which is 2-codable at rate (1:2). First, fuse all but four of the states to get:

$$A_4(D)=\begin{array}{c}0_1\\1_1\\2_1\\3\end{array}\left(\begin{array}{cccc}D^6 & D^3+D^7 & D^8+D^{3+3}+D^{6+4}+D^{7+3} & D^{3+4}+D^{6+5}+D^{7+4}+D^{8+3}+D^{6+4+3}+D^{3+3+3}+D^{7+3+3}\\ D^5 & D^6 & D^7+D^{5+4}+D^{6+3} & D^8+D^{5+5}+D^{6+4}+D^{7+3}+D^{5+4+3}+D^{6+3+3}\\ D^4 & D^5 & D^6+D^{4+4}+D^{5+3} & D^7+D^{4+5}+D^{5+4}+D^{6+3}+D^{4+4+3}+D^{5+3+3}\\ D^3 & D^4 & D^5+D^{3+4}+D^{4+3} & D^6+D^{3+5}+D^{4+4}+D^{5+3}+D^{3+4+3}+D^{4+3+3}\end{array}\right)$$

$$=\begin{array}{c|cccc} & 0_1 & 1_1 & 2_1 & 3\\ \hline 0_1 & D^6 & D^3+D^7 & D^6+D^8+2D^{10} & D^7+D^9+3D^{11}+2D^{13}\\ 1_1 & D^5 & D^6 & D^7+2D^9 & D^8+3D^{10}+2D^{12}\\ 2_1 & D^4 & D^5 & D^6+2D^8 & D^7+3D^9+2D^{11}\\ 3 & D^3 & D^4 & D^5+2D^7 & D^6+3D^8+2D^{10}\end{array}$$

with the resulting eigenvector

$$x=\begin{pmatrix}8\\4\lambda\\8\\8\lambda\end{pmatrix}=\begin{pmatrix}1\cdot 2^3\\1\cdot 2^{5/2}\\1\cdot 2^3\\1\cdot 2^{7/2}\end{pmatrix}.$$

The corresponding induced vector y is again the all 1's vector with corresponding induced input matrix

$$A_4^*(D)=\begin{array}{c|cccc} & 0_1 & 1_1 & 2_1 & 3\\ \hline 0_1 & D^3 & D^2+D^4 & D^3+D^4+2D^5 & D^3+D^4+3D^5+2D^6\\ 1_1 & D^2 & D^3 & D^3+2D^4 & D^3+3D^4+2D^5\\ 2_1 & D^2 & D^3 & D^3+2D^4 & D^3+3D^4+2D^5\\ 3 & D^2 & D^3 & D^3+2D^4 & D^3+3D^4+2D^5\end{array}.$$

Note that indeed $A_4^*(2^{-1})y=y$ (i.e., each row sum of $A_4^*(2^{-1})$ is 1.) Thus, using the lengths from the matrix $A_4^*(D)$, we choose a complete prefix-free list of binary words at each state. From the labeled graph corresponding to $A_4(D)$ we can get a list of codewords at each state. Assignments between these lists at each state gives an encoder. Specifically, one encodes by assigning the words of a binary prefix-free list with lengths 3,2,4,3,4,5,5,3,4,5,5,5,6,6 to the words generated at state 0_1 and a binary prefix-free list with lengths 2,3,3,4,4,3,4,4,4,5,5 to the words generated at the other states. The input lengths and output codewords are shown in Table I. One can modify the encoder by shifting labels and merging states as in the previous example; this is not done here.

As mentioned in Section III, the ARC does not have finite memory. One can show that the MOD-ARC also does not have finite memory. So, the encoder just constructed need not have a sliding-block decoder. In fact, it does not. Thus, if used with a noisy channel, errors may be propagated without bound by the decoder. However, with two additional pieces of information, the state of the MOD-ARC graph can be correctly identified and therefore the state of the encoder graph can be determined, producing a decoder with limited error-propagation. Namely, if time modulo 4 (i.e., $\Sigma_{i=0}^{n}T_i$ modulo 4) is known and polarity ($+1$ or -1) of the transition (equivalently, n modulo 2), then the state of the MOD-ARC can be determined from the Table II (assuming the sequence has polarity $+1$ at time 0).

In the setting of a recording channel, both time modulo 4 and polarity are physically measurable quantities: time modulo 4 is determined by a clock and polarity can be observed by the detector. One can represent this mathematically by modifying the MOD-ARC to obtain a new constraint which takes into account the additional clock/polarity information. This is left to the reader.

Acknowledgment

The authors thank Mignon Belongie for proofreading the paper and for pointing out that the values m_i, m_j in (8) need not be multiples of p, thereby simplifying the construction.

V. Appendix

Proof of Theorem 1: We prove the theorem for the case $k<\infty$. The case $k=\infty$ is left to the reader.

Notation: The symbols $\Sigma_{d,k}$ and $\Sigma_{d,k,a,b}$ denote the (d,k) and (d,k,a,b) constrained systems, respectively. For a random variable T, $H(T)$ denotes the entropy of T and $E(T)$ denotes the expected value of T. Let $C(\Sigma)$ denote the capacity of a constrained system. As defined in the paragraph preceding the statement of Theorem 1, $\log(\lambda^*)$ denotes the capacity of the (d,k) constraint and a^* denotes the average runlength of the ensemble of the set of runlength sequences that satisfy the (d,k) constraint.

TABLE I
CODEWORDS FOR $(d,k,a,b)=(2,7,6,3)$ ARC

	0_1	1_1	2_1	3
0_1	3/000001	2/001 4/0000001	3/001001 4/00000001 5/0000010001 5/0000001001	3/0010001 4/001001001 5/00000010001 5/00000001001 5/00000100001 6/0000010001001 6/0000001001001
1_1	2/00001	3/000001	3/0000001 4/000010001 4/000001001	3/00000001 4/0000100001 4/0000010001 4/0000001001 5/000010001001 5/000001001001
2_1	2/0001	3/00001	3/000001 4/00010001 4/00001001	3/0000001 4/000100001 4/000010001 4/000001001 5/00010001001 5/00001001001
3	2/001	3/0001	3/00001 4/0010001 4/0001001	3/000001 4/00100001 4/00010001 4/00001001 5/0010001001 5/0001001001

TABLE II
STATE INFORMATION FOR $(d,k,a,\ b)=(2,7,6,3)$ ARC DECODER

Time Modulo 4	Polarity (n Modulo 2)	MOD-ARC State
0	0	0
1	0	1
2	0	2
3	0	3
0	1	2
1	1	3
2	1	0
3	1	1

For $0<\lambda<\infty$, define the random variable T^λ by

$$\Pr(T^\lambda=j)=\frac{\lambda^{-j}}{c(\lambda)},\qquad j=d+1,\cdots,k+1,$$

where

$$c(\lambda)=\sum_{j=d+1}^{k+1}\lambda^{-j},$$

$$a(\lambda)=E(T^\lambda)=\sum_{j=d+1}^{k+1}j\lambda^{-j}/c(\lambda).$$

Equivalently, T^λ is the geometrically distributed random variable assuming values $d+1,\cdots,k+1$ with mean $E(T^\lambda)=a(\lambda)$. It is left to the reader to verify that

$$h(\lambda)\equiv H(T^\lambda)=a\log(\lambda)+\log(c),$$

$$\frac{h(\lambda)}{a(\lambda)}=\log(\lambda)+\frac{1}{a}\log(c),$$

$$\lim_{\lambda\to 0}c(\lambda)=+\infty,\qquad \lim_{\lambda\to\infty}c(\lambda)=0,$$

$$\lim_{\lambda\to 0}a(\lambda)=k+1,\qquad \lim_{\lambda\to\infty}a(\lambda)=d+1.\tag{12}$$

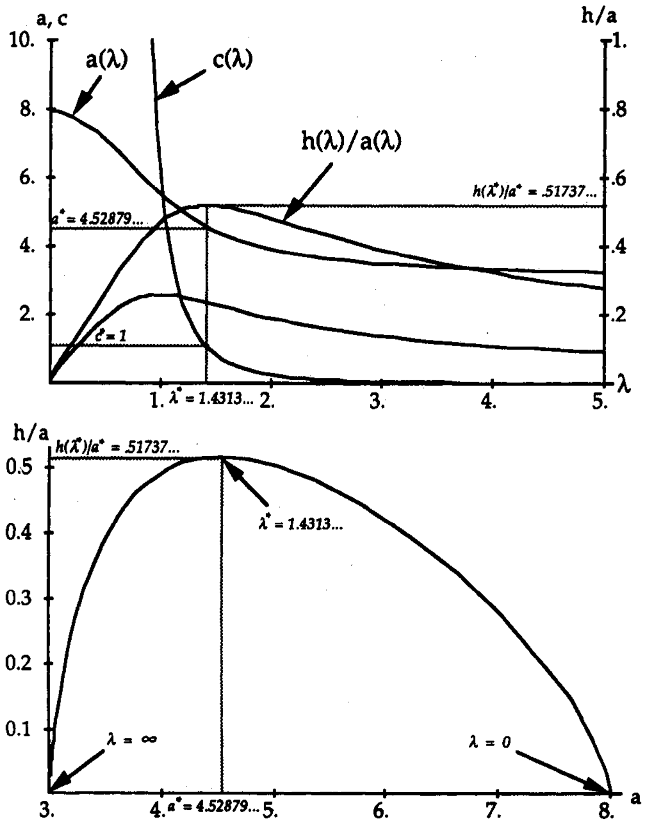

Fig. 15. Mean and entropy-to-mean ratio for $(d,k,a,b)=(2,7,a,\infty)$ ARC.

Also, it is not difficult to show that both $c(\lambda)$ and $a(\lambda)$ are monotonically decreasing in λ, and $h(\lambda)$ is a concave function with maximum at $\lambda=1$ ($h(1)=\log(k-d+1)$, $c(1)=k-d+1$ and $a(1)=(k+d)/2+1$) and $h(\lambda)/a(\lambda)$ is a concave function with maximum $h(\lambda^*)/a^*=\log(\lambda^*)$ where $\lambda^*\ge 1$ is the unique solution to the equation $c(\lambda^*)=1$ ($a(\lambda^*)=a^*$) (see Fig. 15).

It is to be shown that for $d+1<a_0\le a^*$

$$\lim_{b\to\infty}C(\Sigma_{d,k,a_0,b})=\frac{H(T^{\lambda_0})}{a_0},\tag{13}$$

where $\lambda_0\ge 1$ is the unique solution to the equation $a(\lambda_0)=a_0$. This is shown in two steps: 1) if $a_0\le a^*$, then for all b

$$C(\Sigma_{d,k,a_0,b})\le\frac{H(T^{\lambda_0})}{a_0}\tag{14}$$

and 2) for all $d+1<a_0<k+1$

$$\lim_{b\to\infty}C(\Sigma_{d,k,a_0,b})\ge\frac{H(T^{\lambda_0})}{a_0}.\tag{15}$$

This, together with (12) and a little algebraic manipulation, establishes Theorem 1b). For Theorem 1a), simply observe that, for $a_0 > a^*$,

$$\begin{aligned}\log(\lambda^*) &= C(\Sigma_{d,k})\\ &\geq \lim_{b\to\infty} C(\Sigma_{d,k,a_0,b})\\ &\geq \lim_{b\to\infty} C(\Sigma_{d,k,a^*,b})\\ &= \frac{H(T^{\lambda^*})}{a^*} \qquad (\text{by } (13))\\ &= \frac{\log c(\lambda^*)}{a^*} + \log\lambda^* \qquad (\text{by } (12))\\ &= \log\lambda^*,\end{aligned}$$

since, by definition $a(\lambda^*) = a^*$ and $c(\lambda^*) = 1$. Thus,

$$\lim_{b\to\infty} C(\Sigma_{d,k,a,b}) = \log(\lambda^*)$$

as desired.

For the proof of inequality (14), the following lemma is needed [9].

Lemma 4: For $\gamma > 0$ and positive function $f: \mathscr{T} \to \Re^+$, the bound

$$\frac{H(X)+\log(\gamma)}{Ef(X)} \leq \log(\lambda)$$

holds with equality if and only if $\Pr(X = x) = \gamma\lambda^{-f(x)}$ where

$$\sum_{x\in\mathscr{X}} \gamma\lambda^{-f(x)} = 1.$$

Note that this last equation uniquely determines $\lambda > 0$.

Corollary 1: For a positive random variable, $T \in \mathscr{T}$,

$$\frac{H(T)}{ET} \leq \log(\lambda)$$

where

$$\sum_{t\in\mathscr{T}} \lambda^{-t} = 1.$$

Inequality (14) will follow by combining the monotonicity of $a(\lambda)$ and the concavity of $h(\lambda)/a(\lambda)$ (i.e., for $a_0 \leq a^*$, $h(\lambda_0)/a(\lambda_0)$ is monotonically increasing as a_0 increases to a^*) and Corollary 1.

Let $X = (X_1, X_2, \cdots)$ denote the stationary binary process of maximal entropy on $\Sigma_{d,k,a_0,b}$ and let $T = (T_1, T_2, \cdots)$ denote the stationary runlength process induced by X. Let $H_\infty(X)$ and $H_\infty(T)$ denote the entropy of X and T, respectively (i.e., $H_\infty(X) = \lim_{n\to\infty} H_n(X)$, $H_n(X) = (1/n)H(X_1, X_2, \cdots, X_n)$, etc.). Then,

$$C(\Sigma_{d,k,a_0,b}) = H_\infty(X) = \frac{H_\infty(T)}{E(T)} \leq \frac{H_1(T)}{E(T)} = \frac{H(T_1)}{E(T_1)},$$

where $E(T) = E(T_j)$. Since the process X is supported on $\Sigma_{d,k,a,b}$, the ergodic theorem implies that $E(T_1) = a_1 \leq a_0 \leq a^*$. By Corollary 1 and monotonicity

$$\frac{H(T_1)}{E(T_1)} \leq \frac{H(T^{\lambda_1})}{a_1} \leq \frac{H(T^{\lambda_0})}{a_0},$$

where $a(\lambda_1) = a_1$ and so (14) holds as desired.

We now establish (15). (The approach here follows a similar argument in [14].) For each positive integer n and $\epsilon > 0$, define

$$\mathscr{R} = \mathscr{R}_{n,\epsilon} = \left\{ \boldsymbol{r} = r_1 \cdots r_n | d+1 \leq r_i \leq k+1, \sum_{i=1}^{n} (r_i - a_0) < n\epsilon \right\}.$$

View $\mathscr{R}$ as a set of runlength strings. Since $E(T^{\lambda_0}) = a_0$, we can apply the weak law of large numbers and the asymptotic equipartition property to the random variable T^{λ_0} and obtain: for all $\epsilon > 0$, there exists an n such that

$$|\mathscr{R}| \geq 2^{n(H(T^{\lambda_0})-\epsilon)} \tag{16}$$

and

$$\frac{1}{n} < \epsilon \tag{17}$$

(condition (17) is just a convenient technicality that will be used next). Fix $\epsilon > 0$ and such an n, and let $l = \lceil \epsilon n/(a_0 - (d+1)) \rceil$. Let s denote l consecutive runs of $d+1$. Let

$$\mathscr{R}^* = \{\boldsymbol{r} * \boldsymbol{s} | \boldsymbol{r} \in \mathscr{R}\}$$

where the asterisk is the concatenation operator. Note that if $\boldsymbol{u} = u_0 \cdots u_{n+l-1} \in \mathscr{R}^*$, then

$$\sum_{i=1}^{n+l} (u_i - a_0) < 0.$$

Now let $\mathscr{B}$ denote the set of all binary strings that correspond to runlength strings of R^*. Let Λ denote the binary constrained system defined by all possible concatenations of elements of $\mathscr{B}$. For sufficiently large b, depending on n and ϵ, $\Sigma_{d,k,a_0,b}$ contains Λ, and thus

$$C(\Sigma_{d,k,a_0,b}) \geq C(\Lambda).$$

Also, letting $m(\mathscr{B})$ denote the length of the largest string in B,

$$\begin{aligned} C(\Lambda) &\geq \frac{\log(|\mathscr{B}|)}{m(\mathscr{B})}\\ &\geq \frac{\log(2^{n(H(T^{\lambda_0})-\epsilon)})}{n(a_0+\epsilon) + \left(\frac{\epsilon n}{a_0-(d+1)} + 1\right)}\\ &= \frac{H(T^{\lambda_0}) - \epsilon}{a_0 + 2\epsilon + \frac{\epsilon}{a_0-(d+1)}}\end{aligned}$$

by (16) and (17).

Thus, for sufficiently large b,

$$C(\Sigma_{d,k,a_0,b}) \geq \frac{H(T^{\lambda_0})-\epsilon}{a_0+2\epsilon+\dfrac{\epsilon}{a_0-(d+1)}}.$$

But since ϵ was arbitrary and $C(\Sigma_{d,k,a_0,b})$ is monotonically increasing in b,

$$\lim_{b\to\infty} C(\Sigma_{d,k,a_0,b}) \geq \frac{H(T^{\lambda_0})}{a_0}$$

as desired. □

Proof of Lemma 4: The proof consists of two simple steps.

First, let X^* be the random variable with the distribution $q(x)=\gamma\lambda^{-f(x)}$. Then, by definition,

$$\begin{aligned} H(x) &= -\sum_{\mathscr{X}} q(x)\log(q(x)) \\ &= \sum_{\mathscr{X}} q(x)(f(x)\log(\lambda)-\log(\gamma)) \\ &= Ef(X)\log(\lambda)-\log(\gamma). \end{aligned} \tag{18}$$

The *discrimination* (or *Kullback–Liebler number*) for probability distributions p and q is defined by

$$D(p\|q) = \sum_{\mathscr{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right).$$

(See, for example, [19, p. 107].) It is easy to show that this quantity is nonnegative and equal to zero if and only if $p(x)=q(x)$. Specifically,

$$\begin{aligned} &\sum_{\mathscr{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right) \\ &\geq \log(e)\sum_{\mathscr{X}} p(x)\left(1-\frac{q(x)}{p(x)}\right) \\ &= \log(e)\sum_{\mathscr{X}} p(x)-q(x) = \log(e)(1-1)=0, \end{aligned}$$

where the inequality follows from $\log(y)\geq\log(e)(1-1/y)$ with equality if and only if $y=1$. Thus if X has distribution $p(x)$ and X^* has distribution $q(x)=\gamma\lambda^{-f(x)}$, it follows that

$$\begin{aligned} 0 \leq D(p\|q) &= -h(X)+\sum_{\mathscr{X}} p(x)(\log(\gamma\lambda^{-f(x)})-\log(\gamma)) \\ &= -h(X)+Ef(X)\log(\lambda)-\log(\gamma) \end{aligned}$$

or

$$h(X)+\log(\gamma)\leq Ef(X)\log(\lambda).$$

Since $f(x)$ is a positive function, both $Ef(X)$ and $Ef(X^*)$ are positive and

$$\frac{h(X)+\log(\gamma)}{Ef(X)} \leq \log(\lambda) = \frac{h(X^*)+\log(\gamma)}{Ef(X^*)}.$$

The last equality follows from (18). Note that equality holds only when $p(x)=q(x)=\gamma\lambda^{-f(x)}$. □

Proof of Proposition 1: Let $M=\max_i y_i$ be the maximum component of the induced vector and $\mathscr{S}_{\max}=\{i\in\mathscr{S}: y_i=M\}$ the set of states with the maximum component. There are three cases to consider.

Case 1: For *every state* $i\in\mathscr{S}_{\max}$

$$\sum_{j\in\mathscr{S}_{\max}}\sum_{e\in\mathscr{E}_{i,j}} N^{-l^*(e)}\geq 1.$$

Prune $(\mathscr{G},l)$ of all the states except those in $\mathscr{S}_{\max}$. This yields a subgraph with induced vector $\mathbf{y}=\mathbf{1}$.

Case 2: There exist states $i,j\in\mathscr{S}$ and an edge $e\in\mathscr{E}_{i,j}$ such that $l^*(e)\leq 0$.

In this case, the product $N^{-l^*(e)}y_j$ is a positive integer (since both $N^{-l^*(e)}$ and y_j are). Since x is tight, from the definition (8) of $l^*(e)$,

$$N^{-l^*(e)}y_j < y_i.$$

Define

$$u = N^{-l^*(e)}y_j$$

and

$$v = y_i - N^{-l^*(e)}y_j.$$

Then u and v are positive integers that sum to y_i, thereby satisfying condition (11b). Now, let

$$R^*_{i_1}(D) = (0,\cdots,0,D^{l^*(e)},0,\cdots,0),$$

where the single nonzero component is in the jth position, and let

$$R^*_{i_2}(D) = R^*_i(D) - R^*_{i_1}(D).$$

To see that (11a) holds, note that by definition

$$R^*_{i_1}(N^{-1})\mathbf{y} = N^{-l^*(e)}y_j \geq u$$

(in fact equality holds) and, since $\mathbf{y}$ is an N-approximate eigenvector for the input matrix $A^*(D)$,

$$\begin{aligned} R^*_{i_2}(N^{-1})\mathbf{y} &= R^*_i(N^{-1})\mathbf{y} - R^*_{i_1}(N^{-1})\mathbf{y} \\ &\geq y_i - N^{-l^*(e)}y_j = v. \end{aligned}$$

Case 3: For each edge $e\in\mathscr{E}$, $l^*(e)>0$ and there exists a state $i\in\mathscr{S}_{\max}$ such that $\sum_{j\in\mathscr{S}_{\max}}\sum_{e\in\mathscr{E}_{i,j}} N^{-l^*(e)}<1$.

Apply [7, Lemma 7] to show the existence of the required decomposition. □

References

[1] C. Shannon, "The mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423; 623–656, 1948.

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

[3] P. Franaszek, "Run-length-limited variable length coding with error propagation limitation," U.S. Patent 3,689,899, 1972.

[4] ——, "Construction of bounded delay codes for discrete noiseless channels," *IBM J. Res. and Dev.*, vol. 26, no. 4, pp. 506–514, July 1982.

[5] A. M. Patel, "Zero-modulation encoding in magnetic recording," *IBM J. Res. Dev.*, vol. 19, pp. 366–378, July 1975.

[6] B. Marcus, "Factors and extensions of full shifts," *Monats. Math.*, vol. 88, pp. 239–247, 1979.

[7] R. Adler, J. Friedman, B. Kitchens, and B. H. Marcus, "State splitting for variable-length graphs," *IEEE Trans. Inform. Theory*, vol. IT-32, no. 1, pp. 108–113, Jan. 1986.

[8] E. Zehavi and J. K. Wolf, "On run-length codes," *IEEE Trans. Inform. Theory*, vol. IT-34, no. 1, pp. 45–54, Jan. 1988.

[9] C. D. Heegard, "A pair of Information theoretic lemmas with application to runlength coding," 25th Ann. Allerton Conf. Commun., Contr., and Comput., Allerton, IL, Sept. 30–Oct. 2, 1987.

[10] G. S. Dixon, C. A. French and J. K. Wolf, "Results involving (D, K) constrained M-ary codes," *IEEE Trans. Magn.*, vol. MAG-23, pp. 3678–3680, 1987.

[11] A. Gallopoulos, C. D. Heegard, and P. Siegel, "The power spectrum of runlength-limited codes," *IEEE Trans. Commun.*, vol. COM-37, no. 9, pp. 906–917, Sept. 1989.

[12] I. Csiszar, T. Cover, and B.-S. Choi, "Conditional limiting theorems under Markov conditioning," *IEEE Trans. Inform. Theory*, vol. IT-33, no. 6, pp. 788–801, Nov. 1987.

[13] J. Justesen and T. Hoholdt, "Maxentropic Markov chains," *IEEE Trans. Inform. Theory*, vol. 30, no. 4, pp. 665–667, July 1984.

[14] R. Karabed, A. Khayrallah, and D. Neuhoff, "The capacity of costly noiseless channels," IBM Res. Rep. RJ 6040, Jan. 1988.

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

[16] J. Ashley and P. Siegel, "A note on the Shannon capacity of run-length-limited codes," *IEEE Trans. Inform. Theory*, vol. 33, no. 4, pp. 601–605, July 1987.

[17] F. Gantmacher, *The Theory of Matrices*, vol. II. New York: Chelsea, 1959.

[18] R. Karabed and B. Marcus, "Sliding block coding for input restricted channels," *IEEE Trans. Inform. Theory*, vol. 34, no. 2, pp. 2–26, Jan. 1988.

[19] R. Blahut, *Principles and Practice of Information Theory*. Reading, MA: Addison-Wesley, 1972.