Complexity and Sliding-Block Decodability

Jonathan J. Ashley, *Member, IEEE*, Razmik Karabed, and Paul H. Siegel, *Senior Member, IEEE*

***Abstract*—A constrained system, or sofic system, S is the set of symbol strings generated by the finite-length paths through a finite labeled, directed graph. Karabed and Marcus, extending the work of Adler, Coppersmith, and Hassner, used the technique of state-splitting to prove the existence of a noncatastrophic, rate $p:q$ finite-state encoder from binary data into S for any input word length p and codeword length q satisfying $p/q \leq \text{cap}(S)$, the Shannon capacity. For constrained systems that are almost-finite-type, they further proved the existence of encoders enjoying a stronger form of decodability—namely, sliding-block decodability. In particular, their result implies the existence of a 100% efficient (rate $1/2$), sliding-block code for the charge-constrained, runlength-limited constraint with parameters $(d, k; c) = (1, 3; 3)$, an almost-finite-type system with capacity precisely $1/2$. In this paper, we describe two quite different constructions of such codes. The constructions highlight connections between the problem of determining sliding-block decodability of a finite-state encoder and certain problems of colorability for graphs and sets. Using these connections, we show that the problem of determining the existence of a block-decodable input tag assignment for a given rate $p{:}q$, finite-state encoder is NP-complete, for $p > 1$. We also prove NP-completeness results for several related problems in combinatorics and coding.**

***Index Terms*—Complexity, sliding-block decoder, constrained system, NP-complete, graph coloring.**

I. Introduction and Background

CONSTRAINED codes, often called modulation codes, play an important role in the storage and transmission of digital information. The idea is to encode arbitrary user messages to a constrained set of messages which determine the signals that are actually recorded. In the most general terms, the purpose of a constrained code is to improve the performance of the system by matching the characteristics of the recorded signals to those of the channel. There are two classes of constraints over the binary alphabet $\mathcal{A} = \{0, 1\}$ that have found wide use in digital recording systems. The class of *runlength-limited (RLL) constraints* are characterized by two parameters usually denoted by the pair (d, k). The parameter d represents the minimum number of 0's separating consecutive 1's, while k represents the maximum number. The (d, k) constrained system is concisely described by the labeled graph in Fig. 1.

Manuscript received January 26, 1996; revised July 15, 1996.

J. J. Ashley is with IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA.

R. Karabed was with IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA. He is now with GEC Plessey Semiconductors, Inc., Scotts Valley, CA 95067 USA.

P. H. Siegel was with IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA. He is now with the University of California at San Diego, La Jolla, CA 92093 USA.

Publisher Item Identifier S 0018-9448(96)07595-5.

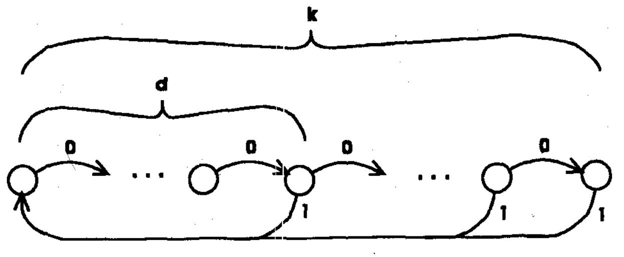

Fig. 1. RLL (d, k) constrained system.

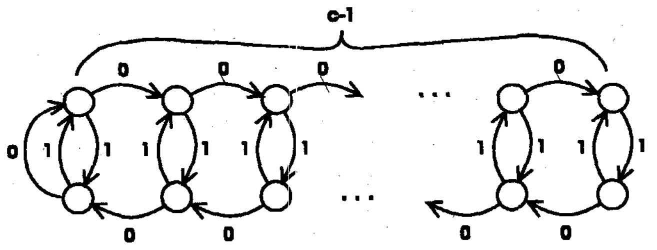

Fig. 2. Charge-constrained system.

The other class of constraints are called *spectral-null constraints* or *charge constraints* and are characterized by a parameter B. The sequences $\boldsymbol{b} = b_1, b_2, \cdots$ in the system satisfy the inequality:

$$\left| \sum_{i=m}^{m+N} (-1)^{\left(\sum_{j=1}^{i} b_j\right)} \right| \leq B$$

for all $m, N \geq 1$. This condition ensures that the constrained sequences have no spectral content at zero frequency (at DC), and codes satisfying such a constraint are often referred to as *DC-free*.

Constrained systems that combine the RLL and charge constraints are sometimes described by the parameters $(d, k; c)$, where d and k are as above, and the parameter c corresponds to setting $B = 2c$ in the running digital sum bound in the inequality above. A labeled graph describing the charge-constrained sequences for $B = 2c$ is shown in Fig. 2.

Shift-invariant sets of sequences, as illustrated by the RLL and spectral-null constraints just described, have been studied in information theory, automata theory, and symbolic dynamics, where they are called discrete noiseless channels, regular languages, and sofic systems, respectively. The fundamental problem is to design efficient, invertible, finite-state encoders from binary data to sequences satisfying the constraint. The efficiency is measured by comparing the encoder rate p/q to the Shannon capacity of the constraint which represents an upper bound on the achievable rate [15]. Since the pioneering work of Shannon, there has been tremendous progress in tackling this design problem. We now summarize the key concepts and results in constrained coding that underlie and motivate this paper.

A. Preliminaries on Constrained Systems and Encoders

A *labeled graph* $G = (V, E, L)$ consists of a finite set of states $V = V_G$; a set $E = E_G$ of directed edges, each edge e having initial state $\sigma(e)$ in V and terminal state $\tau(e)$ in V_G; and an edge label function $L: E_G \rightarrow \Sigma$, where Σ is a finite symbol alphabet. The *adjacency matrix of* G, A_G, is the $|V_G| \times |V_G|$ matrix that describes the underlying directed graph, and is defined by

$$A_G = \{a_{i,j}\}_{i,j}, \quad 1 \leq i, j \leq |V_G|$$

where $a_{i,j}$ is the number of edges from state i to state j. We will sometimes refer to G as simply a *graph* when use of this term is not ambiguous.

The qth *power of* G, denoted G^q, is the labeled graph with state set $V = V_G$, a directed edge from state u to state v for every path of length q in G, and the corresponding edge labels of length q. We will also make use of the qth *higher edge graph of* G, denoted $G^{[q]}$, which has a state for every path of length $q-1$ in G, and an edge for each path of length q in G. The edge corresponding to the path $e_1 e_2 \cdots e_q$ has initial state $e_1 \cdots e_{q-1}$, terminal state $e_2 \cdots e_q$, and edge label given by the length-q string generated by the path in G. Another useful object is the *fiber product* $G * H$ of two labeled graphs, G and H. The vertex set is given by

$$V_{G*H} = V_G \times V_H = \{(u, u') | u \in V_G, u' \in V_H\}$$

and an edge $(u, u') \xrightarrow{a} (v, v')$ belongs to E_{G*H} if and only if $u \xrightarrow{a} v \in E_G$ and $u' \xrightarrow{a} v' \in E_H$.

A *constrained system* $S = S(G)$ is the set of finite sequences generated by sequentially reading the labels from edges in paths in a labeled graph G. The labeled graph G is said to be a *presentation* of the system S. The constrained system S is sometimes referred to as a *constraint*. The systems corresponding to G^q and $G^{[q]}$ are denoted S^q and $S^{[q]}$, respectively. Note that the fiber product generates the intersection of the constraints $S(G)$ and $S(H)$, that is, $S(G * H) = S(G) \cap S(H)$. We will be concerned largely with *irreducible constraints*, meaning systems S that can be presented by a labeled graph G in which the underlying graph is irreducible (strongly connected).

A labeled graph G is said to be *deterministic* when, at each state, the outgoing edges are distinctly labeled. Every constraint has a deterministic presentation. More generally, a graph G has *finite anticipation* a if any pair of paths $e_1, \cdots, e_{a+1}$ and $e'_1, \cdots, e'_{a+1}$ of length $a+1$ with the same initial state and generating the same sequence of labels must have the same first edge, that is, $e_1 = e'_1$. A deterministic graph has anticipation $a = 0$. Finally, G is *lossless* if any two paths with the same initial state and final state generate distinct label sequences.

The capacity of the system S, denoted $\mathbf{cap}(S)$, is the growth rate of the number $N(l; S)$ of sequences of length l in S. More precisely,

$$\mathbf{cap}(S) = \lim_{l \rightarrow \infty} \log N(l; s)/l.$$

If S is irreducible, and G is a lossless presentation of S, the capacity is given by the simple formula

$$\mathbf{cap}_\alpha(S) = \log_\alpha \lambda(A_G)$$

where $\lambda(A_G)$ is the largest eigenvalue of A_G (which is guaranteed to be real and positive by the Perron–Frobenius Theorem). Since $A_{G^q} = A_G^q$, it follows that $\mathbf{cap}(S^q) = q\,\mathbf{cap}(S)$.

Given a constraint S and a positive integer n, we define an (S, n) *encoder* as a lossless, labeled graph $\mathcal{E}$ in which each state has n outgoing edges and the corresponding constrained system satisfies $S(\mathcal{E}) \subseteq S$. A *tagged* (S, n) *encoder* is obtained by assigning distinct *input tags*, drawn from an alphabet of size n, to the n outgoing edges from each state in $V_{\mathcal{E}}$. If an (S, n) encoder has only one state, it is called a *block encoder*. A tagged (S^q, n^p) encoder—where the input tags are the length-p, n-ary strings—will be referred to as a *rate* p:q *finite-state* n*-ary encoder for* S.

Shannon [15] proved that the existence of a rate $kp:kq$ block encoder, for some positive integer k, implies that $p/q \leq \mathbf{cap}(S)$. Conversely, he gave a nonconstructive proof that there exists a rate $kp:kq$ block encoder, for some k, when $p/q < \mathbf{cap}(S)$.

Adler, Coppersmith, and Hassner [1], using the technique of *state-splitting* developed by Marcus [10] in the context of symbolic dynamics, proved that, for any p, q satisfying $p/q \leq \mathbf{cap}(S)$, there exists a rate $p:q$, finite-state encoder for S with finite anticipation. The latter condition ensures that the encoding is invertible by means of a state-dependent decoder. Moreover, the proof given was constructive. (An exposition of the state-splitting algorithm may be found in [12] and [13].) In this paper, we will be concerned with encoders that satisfy a stronger form of decodability, as we now describe.

B. Stronger Forms of Decodability

Generally, constrained codes are applied in the context of noisy recording or communication channels. The decoder is therefore likely to be operating on a corrupted sequence which may or may not belong to the constraint S. In this setting, a state-dependent decoder may suffer catastrophic failure in response to even a single symbol error. Therefore, practical considerations require the design of encoders with stronger decodability properties.

A tagged, rate p:q encoder for S is *noncatastrophic* if it has finite anticipation and, in addition, whenever the output sequences generated by two right-infinite paths differ in only a finite number of positions, the corresponding input tag sequences also differ in only a finite number of positions. Karabed and Marcus [7] proved that, for any constrained system S and any positive integers p, q satisfying $p/q \leq \mathbf{cap}(S)$, there is a rate $p:q$ noncatastrophic encoder.

An even stronger form of decodability—*sliding-block decodability*—is the most important in applications and is defined as follows. A tagged, rate $p : q$ encoder for S is (m, a)-*sliding-block-decodable* if, for any two paths $e_{-m} \cdots e_0 \cdots e_a$ and $e'_{-m} \cdots e'_0 \cdots e'_a$ that generate the same sequence, the edges e_0 and e'_0 have the same input tag. For such an encoder, the sliding-block decoder $\mathcal{D}$ is given by a mapping from $(\Sigma(S^q))^{m+a+1}$ to the set of length-p, n-ary input tags, where $\Sigma(S)$ denotes the symbol alphabet of the constraint S. The decoder mapping has the property that, if the codeword sequence $c_1c_2 \cdots$ is generated by the encoder from the input tag sequence $b_1b_2 \cdots$, then

$$b_i = \mathcal{D}(c_{i-m}, \cdots, c_i, \cdots, c_{i+a}), \quad \text{for } i > m.$$

This property ensures that a single codeword error at the input to the decoder can corrupt at most $m+a+1$ input tags, thereby limiting the extent of error propagation to a finite duration. An encoder that is $(0, 0)$-sliding-block-decodable is said to be *block-decodable*.

In order to state existence results for sliding-block-decodable encoders, we must first introduce two classes of constrained systems—*finite-type* and *almost-finite-type* systems. A labeled graph G is said to be (m, a)-*definite* if, given any sequence $\boldsymbol{w} = w_{-m} \cdots w_0 \cdots w_a$ in $S(G)$, the set of paths $e_{-m} \cdots e_0 \cdots e_a$ that generate $\boldsymbol{w}$ agree on the edge e_0. A constraint S is *finite-type* if it can be presented by an (m, a)-*definite* graph, for some m and a. Finite-type constraints are also characterized intrinsically by the *finite memory* property: there exists an integer N such that, for any symbol c in the alphabet and any sequence $\boldsymbol{w} = w_1w_2 \cdots w_n$ in S of length $n \geq N$, the sequence $\boldsymbol{w}c$ is in S if and only if the sequence $w_{n-N+1} \cdots w_nc$ is in S. The (d, k) RLL constraints are an important example of finite-type constrained systems. Adler, Coppersmith, and Hassner [1] showed that when their state-splitting algorithm is applied to a definite graph G that presents a finite-type constraint, the (untagged) encoder graph that is generated is also definite. This implies that, with any assignment of input tags, the resulting tagged encoder is (m, a)-sliding-block-decodable for some finite m and a.

It is not difficult to see that the charge-constrained systems are not finite-type. They belong to the class of *almost-finite-type* systems, which can be characterized as systems having a presentation with finite anticipation and finite co-anticipation (the anticipation of the graph obtained by reversing the direction of the edges). Finite-type constraints are a proper subclass of almost-finite-type constraints. Karabed and Marcus [7], building upon Marcus [11], improved upon the result in [1], showing that, for any almost-finite-type constrained system S and any positive integers p, q satisfying $p/q \leq \mathbf{cap}(S)$, there is a rate $p : q$, sliding-block-decodable encoder.

The research leading to the coding theorems in [11] and [7] was motivated, at least in part, by the problem of determining the existence of a sliding-block-decodable encoder with rate equal to the capacity for the charge-constrained, RLL constraint with parameters $(d, k; c) = (1, 3; 3)$. This constraint is almost-finite-type [11], though not finite-type, and has capacity exactly equal to $1/2$. Patel [14] had designed a rate $1/2$ encoder mapping for this constraint, but it required unbounded anticipation. By modifying the encoder, he was able to demonstrate a family of sliding-block-decodable codes

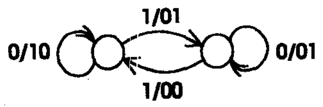

Fig. 3. $(1, 0)$-sliding-block-decodable tagged encoder for $(d, k) = (1, 3)$.

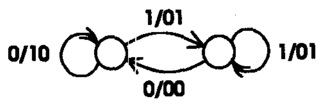

Fig. 4. $(0, 0)$-sliding-block-decodable tagged encoder for $(d, k) = (1, 3)$.

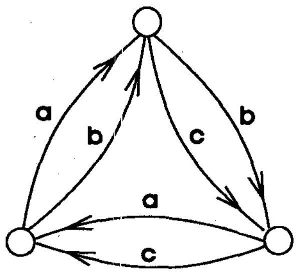

Fig. 5. Encoder graph with no sliding-block-decodable tagging.

with rates approaching from below, but not achieving, rate $1/2$. The main theorem in [7] proves that there are, in fact, rate $1/2$ sliding-block-decodable encoders for the $(1, 3; 3)$ constraint. In this paper, we will describe the first explicit constructions of such encoders.

C. Input Tag Assignments

In view of the existence results for sliding-block-decodable encoders, a problem that arises naturally during the code construction procedure is the choice of input tag assignment to achieve sliding-block decodability and to minimize the length of the sliding-block decoder window.

To illustrate some of the issues involved in input tag assignment, consider the rate $1 : 2$ encoder for the $(d, k) = (1, 3)$ constraint shown in Fig. 3. It is not difficult to see that the encoder is $(1, 0)$-sliding-block-decodable. The alternative assignment shown in Fig. 4, however, is $(0, 0)$-sliding-block-decodable, or block-decodable.

In contrast, the graph in Fig. 5 is an untagged rate $1 : 1$ encoder for the constraint S that it presents. One can show easily that there is, in fact, no input tag assignment that induces block decodability [16], [7]. However, the higher edge graphs $\mathcal{E}^{[q]}$ of the encoder graph in Fig. 5 will also be rate $1 : 1$ encoders for the corresponding higher edge constraints $S^{[q]}$, and there is an input tag assignment in the case $q = 2$ that yields a $(0, 1)$-sliding-block-decodable encoder, as shown in Fig. 6. On the other hand, there is no input tag assignment for any higher edge graph of the encoder graph in Fig. 7 that produces an (m, a)-sliding-block-decodable tagged encoder, for any finite values of m and a [7].

Given a particular choice of assignment, there is an efficient algorithm for testing whether the tagged encoder is (m, a)-sliding-block-decodable, which we outline here [12]. Let $\mathcal{E}$ be a tagged (S, n)-encoder graph. Let $A_{\mathcal{E}*\mathcal{E}}$ be the adjacency

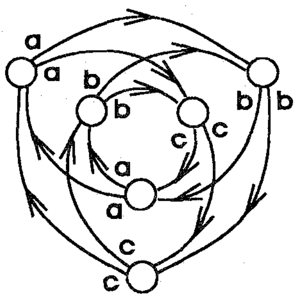

Fig. 6. Edge -graph encoder with $(0, 1)$-sliding-block-decodable tagging.

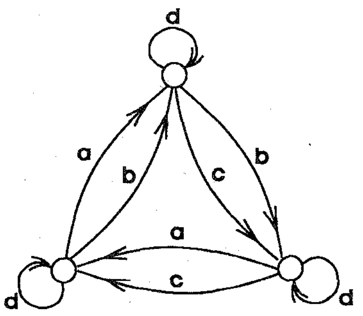

Fig. 7. Encoder with no sliding-block-decodable tagging of any higher edge graph.

matrix of the fiber product $\mathcal{E} * \mathcal{E}$. Define the matrix $B_{\mathcal{E}*\mathcal{E}}$ to be the $|V_{\mathcal{E}}|^2 \times |V_{\mathcal{E}}|^2$ matrix, with rows and columns indexed by the states of $\mathcal{E} * \mathcal{E}$, such that for states u, u', v, v', the entry $B_{\mathcal{E}*\mathcal{E}}((u, u'), (v, v'))$ is equal to the number of pairs of distinct edges $u \rightarrow v$ and $u' \rightarrow v'$ with the same (output) label but distinct input tags. Then, the tagged encoder $\mathcal{E}$ is (m, a)-sliding-block-decodable if and only if

$$A^{m}_{\mathcal{E}*\mathcal{E}} \cdot B_{\mathcal{E}*\mathcal{E}} \cdot A^{a}_{\mathcal{E}*\mathcal{E}} = 0.$$

For an encoder graph with even moderate size, however, it is impractical to apply this test to every possible input tag assignment. Therefore, it would be very desirable to have an efficient algorithm to solve the following problem:

Encoder Input Tag Assignment: Given an (S, n)-encoder graph $\mathcal{E}$, does there exist an input tag assignment such that the tagged encoder is (m, a)-sliding-block-decodable?

In view of the fact that the construction of an (S, n)-encoder graph via state-splitting often generates a graph with out-degree at least (and possibly greater than) n, it is natural to consider the following more general problem:

Subgraph Encoder Input Tag Assignment: Given a labeled graph G having out-degree at least n at every state, does there exist an assignment of input tags to the edges of G such that the tagged (S, n)-encoder obtained by restricting the assignment to an (S, n)-encoder $\mathcal{E} \subseteq G$ is (m, a)-sliding-block-decodable?

We will establish connections between the input tag assignment problems stated above and problems relating to colorability of graphs and sets. The techniques used in the two constructions of 100% efficient, sliding-block codes for the $(1, 3; 3)$ constraint that are presented in this paper draw upon these connections. We will also use the connections to address the complexity of the two input tag assignment problems.

D. Outline of the Remainder

In Section II, we consider the problem of assigning input tags in a consistent manner in order to produce a sliding-block-decodable tagged encoder graph. We first show that the Encoder Input Tag Assignment problem is related to the well-known combinatorial Graph n-Coloring problem. We then show that the more general problem of Subgraph Encoder Input Tag Assignment is related to a combinatorial problem that we will refer to as Set n-Coloring.

In Sections III and IV, we describe in some detail two constructions of rate $1/2$, sliding-block codes for the $(d, k; c) = (1, 3; 3)$ constraint. These code designs represent, to the best of our knowledge, the first 100% efficient sliding-block codes for this constraint, thereby realizing the promise of the theorem of Karabed and Marcus [7].

Specifically, in Section III, motivated by the connections established in Section II, we develop an out-splitting heuristic that we apply to construct a 20-state, rate $4:8$ encoder graph for the $(d, k; c) = (1, 3; 3)$ constraint which, with a specified input tag assignment, is $(3, 5)$-sliding-block-decodable. This code was originally announced in [8].

Section IV describes a new code construction approach based upon a matrix generalization of the state-splitting algorithm. The problem of determining the existence of a $(0, a)$-sliding-block-decodable input tag assignment for a subgraph encoder generated by a rounds of matrix-based out-splittings is recast in terms of a constrained integer programming problem that can, in turn, be interpreted as a Set n-Coloring problem. We then apply the matrix-based state-splitting technique to construct another rate $4:8$ encoder that has a larger number of states than the encoder of Section III, namely 104, but which we endow with an input tag assignment that ensures $(0, 2)$-sliding-block decodability.

Section V addresses computational complexity issues associated with the design of sliding-block-decodable encoders, making use of the connection to coloring problems. We begin the section with a very brief review of the theory of NP-completeness.

In Section V-A, we provide a polynomial-time reduction of the Graph n-Coloring problem to the Encoder Input Tag Assignment problem. Since the Graph n-Coloring problem is known to be NP-complete, this proves that the general problem of determining the existence of an (m, a)-sliding-block-decodable input tag assignment for a given (S, n) encoder is NP-complete. It also follows that even for fixed n, the problem remains NP-complete for $n \geq 3$.

In Section V-B, we polynomially reduce the Satisfiability (SAT) problem—the original NP-complete problem—to the Set n-Coloring problem, proving that the latter is NP-complete. We then reduce Set n-Coloring to the input tag assignment problem, representable as a constrained integer programming problem, that arises in the matrix-based code construction, proving that these are both NP-complete. In

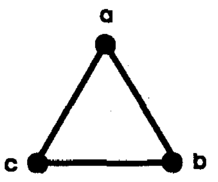

Fig. 8. Edge-coloring graph for the first encoder.

fact, we also conclude that for fixed n, the problems are NP-complete for $n \geq 2$.

Finally, in Section V-C, we consider a natural mapping problem for graph edge systems called 1-Block Surjection. By a reduction of Set 2-Coloring to an instance of this problem, we show that 1-Block Surjection is NP-complete.

II. Coloring Problems and Input Tag Assignment

In this section, we establish an elementary relationship between the Encoder Input Tag Assignment problem and the combinatorial Graph n-Coloring problem. The connection is clarified by the introduction of an object we refer to as the *edge-coloring graph*. We then draw a simple connection between the Subgraph Encoder Input Tag Assignment problem and a more general coloring problem that we call Set n-Coloring.

We first introduce a relation on the edges of a labeled graph (G, L). Two edges e and e' are (m, a)*-adjacent* if there exist two paths

$$e_{-m} \cdots e_{-1}e_0e_1 \cdots e_a$$
$$e'_{-m} \cdots e'_{-1}e'_0e'_1 \cdots e'_a$$

in G, such that $e_0 = e$, $e'_0 = e'$, and

$$L(e_{-m} \cdots e_{-1}e_0e_1 \cdots e_a) = L(e'_{-m} \cdots e'_{-1}e'_0e'_1 \cdots e'_a).$$

It is easy to verify that the transitive closure of (m, a)-adjacency is an equivalence relation. We call this equivalence relation (m, a)*-connectedness* and we denote the equivalence class of e by $[e]$. From the definition, it follows that a tagged encoder $\mathcal{E}$ is (m, a)-sliding-block-decodable if and only if the following two conditions are satisfied:

1) Edges that are (m, a)-connected receive the same input tag.
2) Edges with the same initial state receive distinct input tags.

The (m,a) *edge-coloring graph* $C_G(m, a)$ is defined as follows. Let the vertices V be the set of equivalence classes for the connectedness relation. There is an undirected edge connecting two vertices u and v whenever $u = [e]$ and $v = [f]$, where $e \neq f$ and $\sigma(e) = \sigma(f)$. The coloring graph for the encoder graph in Fig. 5 is shown in Fig. 8, and that for Fig. 6 is shown in Fig. 9.

A well-known combinatorial problem whose complexity has been studied is that of *graph n-colorability*, which we now describe. Let Σ_n denote the alphabet $\{1, 2, \cdots, n\}$. An *n-coloring* of an undirected graph $H = (V_H, E_H)$ is an assignment $P: V_H \to \Sigma_n$ such that $P(u) \neq P(v)$ whenever the states $u, v \in V_H$ are connected by an edge $e \in E_H$. The

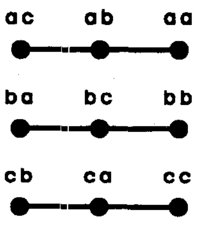

Fig. 9. Edge-coloring graph for the second encoder.

Graph n-Coloring (or chromatic number) problem is specified as follows.

Graph n-Coloring: Given an undirected graph $H = (V_H, E_H)$ and a positive integer n, does there exist an n-coloring of H?

For an (S, n) encoder $\mathcal{E}$, the following theorem identifies the relationship between input tag assignments that are (m, a)-sliding-block-decodable and n-colorings of the (m, a) edge-coloring graph $C_{\mathcal{E}}(m, a)$.

Theorem 2.1: Let $\mathcal{E}$ be an (S, n) encoder, with (m, a) edge-coloring graph $C_{\mathcal{E}}(m, a)$. There is a one-to-one correspondence between n-ary input tag assignments yielding (m, a)-sliding-block-decodable encoders and n-colorings of $C_{\mathcal{E}}(m, a)$.

Proof: Let P be an n-coloring of $C_{\mathcal{E}}(m, a)$. The corresponding input tag assignment $\mathcal{I}: E_{\mathcal{E}} \to \Sigma_n$ is defined by $\mathcal{I}(e) = P([e])$. Since P defines an n-coloring, the n outgoing edges from each state in $\mathcal{E}$ will have distinct input tags, as required. The decoder mapping $\mathcal{D}: (\Sigma(S))^{m+a+1} \to \Sigma_n$ is defined as follows. Let $\boldsymbol{w} = w_{-m} \cdots w_0 \cdots w_a$ be a string in S generated the path $\boldsymbol{e} = e_{-m} \cdots e_0 \cdots e_a$ in $\mathcal{E}$. Denoting e_0 by e, we define $\mathcal{D}(\boldsymbol{w}) = \mathcal{I}(e)$. Note that if $\boldsymbol{w}$ is generated by another path $\boldsymbol{e}' = e'_{-m} \cdots e'_0 \cdots e'_a$ in $\mathcal{E}$, with $e'_0 = e'$, then e and e' are (m, a)-connected, so $[e] = [e']$. Thus

$$\mathcal{I}(e) = P([e]) = P([e']) = \mathcal{I}(e')$$

so the decoder function $\mathcal{D}$ is well-defined. Conversely, suppose that $\mathcal{I}$ is an n-ary input tag assignments yielding (m, a)-sliding-block-decodable encoder $\mathcal{E}$, with decoder $\mathcal{D}$. If e and e' are (m, a)-adjacent, then there are paths $e_{-m} \cdots e_{-1}ee_1 \cdots e_a$ and $e'_{-m} \cdots e'_{-1}e'e'_1 \cdots e'_a$ in $\mathcal{E}$ that generate the same word $\boldsymbol{w}$. So

$$\mathcal{I}(e) = \mathcal{D}(\boldsymbol{w}) = \mathcal{I}(e').$$

Next if f and f' are edges of $\mathcal{E}$ in the same (m, a)-connectedness equivalence class $[f] = [f']$, then there is a chain of such pairs (e, e') "connecting" f to f'. Thus $\mathcal{I}(f) = \mathcal{I}(f')$. Hence we can define a coloring P of the vertices of $C_{\mathcal{E}}(m, a)$ by $P([e]) = \mathcal{I}(e)$. By definition, if $[f]$ and $[f']$ are connected by an edge in $C_{\mathcal{E}}(m, a)$, then there are distinct edges $e \in [f]$ and $e' \in [f']$ having the same initial state $\sigma(e) = \sigma(e')$ and, therefore, distinct input tags, $\mathcal{I}(e) \neq \mathcal{I}(e')$. We conclude that

$$P([f]) = \mathcal{I}(e) \neq \mathcal{I}(e') = P([f']).$$

Thus P is an n-coloring of $C_{\mathcal{E}}(m, a)$. $\square$

If the graph G is not an (S, n) encoder, then the problem of determining whether it supports an (m, a)-sliding-block-decodable (S, n) encoder is related to another combinatorial problem, which we call Set n-Coloring, defined as follows.

Set n-Coloring: Given a collection of finite subsets $S_1, S_2, \cdots, S_m$ of a set U, does there exist a coloring

$$\mathcal{I}: \bigcup_{k=1}^{m} S_k \to \{0, 1, \cdots, n-1\}$$

such that: for each $1 \le k \le m$, and for each color $0 \le j \le n-1$, the set S_k has at least one element of color j?

Theorem 2.2: Let G be a labeled graph presenting the constrained system S. Then the graph G can be labeled with input tags in such a way that a tagged subgraph $\mathcal{E}$ forms an (m, a)-sliding-block-decodable (S, n) encoder if and only if the following two steps can be performed.

1) Solve the following instance of the Set n-Coloring problem. The set U is defined to be the set of vertices of the edge-coloring graph $C_G(m, a)$. For each state s of G, define a subset $S_s \subseteq U$ by

$$S_s = \{u \in U: u = [e] \text{ where } \sigma(e) = s\}.$$

Use the solution $\mathcal{I}: \cup_s S_s \to \{0, 1, \cdots, n-1\}$ to define an input tagging of G by setting $\mathcal{I}(e) = \mathcal{I}([e])$.

2) Define the (S, n)-encoder $\mathcal{E}$ by deleting edges from G so that the remaining graph has, at each state s, and for each input label $0 \le j \le n-1$, exactly one edge e with $\sigma(e) = s$ and $\mathcal{I}(e) = j$.

Proof: One only has to observe that if $e_{-m} \cdots e_0 \cdots e_a$ and $e'_{-m} \cdots e'_0 \cdots e'_a$ are paths in G that generate the same label sequence, then the edges e_0 and e'_0 represent the same vertex u in the set U in the statement of the theorem. Thus an input tagging of G that yields an (m, a)-sliding-block-decodable (S, n) encoder $\mathcal{E} \subseteq G$ produces a set coloring as in the statement of the theorem, and conversely. □

III. A 20-State, Rate 4:8, $(3,5)$-Sliding-Block-Decodable $(1,3;3)$ Encoder

In this section, motivated to some extent by the discussion in the previous section, we develop a heuristic for choosing out-splittings in order to achieve an encoder that, with an appropriate input tag assignment, is sliding-block-decodable. We then describe in full detail the construction of a rate $4:8$, 20-state encoder graph for the $(d, k; c) = (1, 3; 3)$ constraint. We specify a consistent input tag assignment for which the resulting tagged encoder is $(3,5)$-sliding-block-decodable. This code was first announced in [8].

A. A Heuristic for Out-Splitting to Achieve Sliding-Block Decodability

Suppose that we start with a graph $G^{(0)} = G$ presenting a constrained system S, and perform a rounds of out-splitting leading through graphs $G^{(1)}$, $G^{(2)}$, and so on, up to $G^{(a)}$ so that $G^{(a)}$ has out-degree exactly n. We would like to label the edges of $G^{(a)}$ with input symbols so that we obtain an (m, a)-sliding-block-decodable (S, n) encoder.

We will now develop a heuristic to guide the sequence of out-splittings when the objective is to produce such a tagged encoder. We first introduce an auxiliary graph $\boldsymbol{H}(G; m, a)$, defined to be the subgraph of the fiber product $G * G$ of G with itself that is determined by restricting the edge set to edge pairs (e, f) where e and f are (m, a)-adjacent.

We remark that the graph $\boldsymbol{H}(G; m, a)$ has the following easily verified properties:

a) $\boldsymbol{H}(G; 0, 0)$ is equal to the fiber product $G * G$.

b) $\boldsymbol{H}(G; m, a+1)$ may be obtained from $\boldsymbol{H}(G; m, a)$ by first eliminating from $\boldsymbol{H}(G; m, a)$ all edges terminating in states having no outgoing edges and, then, eliminating those states.

c) Similarly, $\boldsymbol{H}(G; m+1, a)$ may be obtained from $\boldsymbol{H}(G; m, a)$ by first eliminating from $\boldsymbol{H}(G; m, a)$ all edges originating in states having no incoming edges and, then, eliminating those states.

These properties yield a recursive procedure for constructing the graph $\boldsymbol{H}(G; m, a)$, beginning with the original fiber product graph $G * G$.

An arbitrary out-splitting of two states s and s' of the graph G, yielding a graph G', induces an out-splitting of the state (s, s') of $\boldsymbol{H}(G; m, a)$ in $G * G$ as follows. For each atom A in the partition of the outgoing edges from s, and for each atom A' in the partition of the outgoing edges from s', we define the subset

$$\{(e, e'): \sigma((e, e')) = (s, s') \text{ and } e \in A \text{ and } e' \in A'\}.$$

The collection of such subsets that are nonempty form a partition of the edges following state (s, s') that we can use to define an out-splitting of (s, s'). Although we do not prove it, the resulting graph is actually $\boldsymbol{H}(G'; m, a)$. We prove a special case as Lemma 3.1.

Suppose that (s, s') is a state of $\boldsymbol{H}(G; m, a)$, and that (e, e') and (f, f') are two edges in $\boldsymbol{H}(G; m, a)$ emanating from state (s, s'). Suppose that we out-split the graph G in such a way that the partition defining the splitting of state s has edges e and f in a single atom (defining a descendant state t in the resulting graph G') while in the partition that defines the splitting of state s', the edges e' and f' are in distinct atoms (defining distinct descendant states $t^{(1)}$ and $t^{(2)}$ in G'). Then in the induced splitting on $\boldsymbol{H}(G; m, a)$, among the descendants of state (s, s') are two states: $(t, t^{(1)})$ and $(t, t^{(2)})$. Among the descendants of any incoming edge (g, g') into state (s, s') are two edges: $(h, h^{(1)})$ terminating at $(t, t^{(1)})$ and $(h, h^{(2)})$ terminating at $(t, t^{(2)})$. Thus the edges $h^{(1)}$ and $h^{(2)}$ of G' represent the same vertex of $C_{G'}(m, a)$, which then has a self-loop since $\sigma(h^{(1)}) = \sigma(h^{(2)})$, so there is no input tagging of G' that defines an (m, a)-sliding-block-decodable encoder. Thus the separation of the edges e' and f' into distinct atoms must eventually be mirrored in subsequent rounds of out-splitting by the separation of (the descendants of) the edges e and f.

This motivates the following heuristic out-splitting condition. Say a round of out-splitting of a labeled graph G is

(m, a)-*label consistent* if for each state (s, s') of $\boldsymbol{H}(G; m, a)$, and for any two edges (e, e') and (f, f') emanating from state (s, s'), the edges e and f are in the same atom of the partition defining the out-splitting of state s if and only if the edges e' and f' are in the same atom of the partition defining the out-splitting of state s'. The next lemma shows that, when we perform a round of (m, i)-label-consistent out-splittings on G, there is a simple procedure for determining the $(m, i+1)$-adjacent pairs of edges in the split graph G' from the (m, i)-adjacent pairs in G.

Lemma 3.1: Suppose that G is a labeled graph and that G' is obtained from G by one round of (m, i)-label-consistent out-splitting. Then the graph $\boldsymbol{H}(G'; m, i+1)$ is obtained from the graph $\boldsymbol{H}(G; m, i)$ in the following two-step operation.

1) Eliminate from $\boldsymbol{H}(G; m, i)$ all states having no outgoing edges, along with their incoming edges. This leaves the graph $\boldsymbol{H}(G; m, i+1)$.
2) Out-split $\boldsymbol{H}(G; m, i+1)$ according to the following partitions. For each state (s, s') of $\boldsymbol{H}(G; m, i+1)$, declare that two edges (e, e') and (f, f') emanating from state (s, s') be in the same atom of the partition of its outgoing edges if and only if the edges e and f are in the same atom of the partition that defines the out-splitting of state s in G (which happens if and only if the edges e' and f' are in the same atom of the partition that defines the out-splitting of state s' in G).

Proof: Step 1) is simply a restatement of property b) above.

To prove that step 2) yields the desired graph, we first introduce some notation. Suppose K' is a graph obtained by out-splitting a graph K. Any edge e of K belongs to a unique atom of the partition used to define the splitting of the state $\sigma(e)$. Denote by $\alpha(e)$ the state of K' corresponding to this atom. If a state t of K' results from out-splitting state s of K, then for each edge e of K with $\tau(e) = s$, there is a unique edge g of K' with $\tau(g) = t$ that maps onto e by the graph homomorphism from K' to K defined by the out-splitting. We denote $g = \langle e, t\rangle$.

Now let H' be the graph obtained by out-splitting $\boldsymbol{H}(G; m, i+1)$ according to the partitions defined in 2). We must show that H' and $\boldsymbol{H}(G'; m, i+1)$ are isomorphic as labeled graphs. Define a mapping Φ from the states of $\boldsymbol{H}(G'; m, i+1)$ to the states of H' as follows. Let (t, t') be a state of $\boldsymbol{H}(G'; m, i+1)$. The state t of G' is the result of out-splitting some state s (its *parent*) in G. Similarly, t' has a parent s'. Any label sequence $L(g_{-m} \cdots g_{-1}g_0g_1 \cdots g_{i+1})$ generated by a path in G' with $\tau(g_{-1}) = t$ is also generated by a path $e_{-m} \cdots e_{-1}e_0e_1 \cdots e_{i+1}$ in G with $\tau(e_{-1}) = s$. The same can be said of t' and its parent s'. Thus (s, s') is a state of $\boldsymbol{H}(G; m, i+1)$. Because the out-splitting leading from G to G' is (m, i)-label-consistent, the state $\alpha((e, e'))$ of H' is independent of the choice of edges e and e' for which $\alpha(e) = t$, $\alpha(e') = t'$, and $\sigma((e, e')) = (s, s')$. Thus we can define $\Phi((t, t')) = \alpha((e, e'))$. It also follows from the fact that the out-splitting from G to G' is (m, i)-label-consistent that the state $\alpha((e, e'))$ of H' uniquely determines the states $\alpha(e)$ and $\alpha(e')$ of G'. Thus Φ is a one-to-one map.

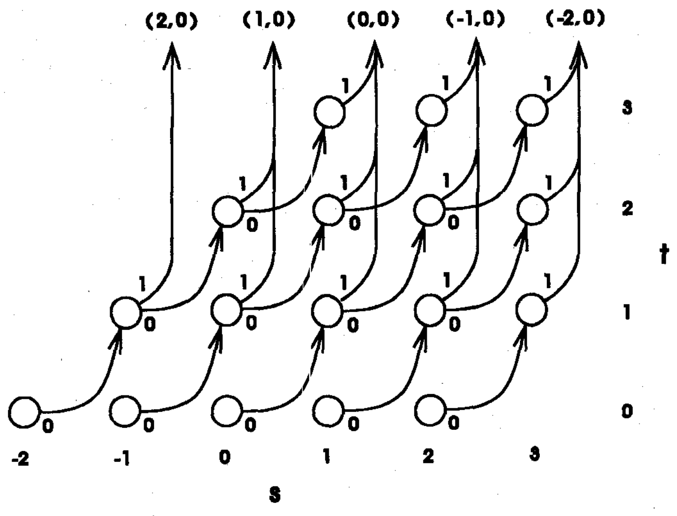

Fig. 10. Shannon cover of $(d, k; c) = (1, 3; 3)$ system.

Define a mapping Ψ from the edges of $\boldsymbol{H}(G'; m, i+1)$ to the edges of H' as follows. Any edge of $\boldsymbol{H}(G'; m, i+1)$ preceding the state (t, t') can be expressed as $(\langle e, t\rangle, \langle e', t'\rangle)$, where e is an edge of G preceding the parent state s of the state t, e' is an edge of G preceding the parent state s' of the state t', and (e, e') is an edge preceding the state (s, s') of $\boldsymbol{H}(G; m, i+1)$. Define

$$\Psi((\langle e, t\rangle, \langle e', t'\rangle)) = \langle (e, e'), \Phi((t, t'))\rangle.$$

It follows easily from the fact that Φ is one-to-one that Ψ is one-to-one as well. It remains to show that the pair of maps (Φ, Ψ) together define a graph homomorphism from $\boldsymbol{H}(G'; m, i+1)$ to H'. It follows directly from the definition of Ψ that $\tau \circ \Psi = \Phi \circ \tau$. The fact that $\sigma \circ \Psi = \Phi \circ \sigma$ follows from

$$\begin{aligned}\Phi \circ \sigma((\langle e, t\rangle, \langle e', t'\rangle)) &= \Phi((\alpha(e), \alpha(e')))\\ &= \alpha((e, e'))\\ &= \sigma(\langle (e, e'), \Phi((t, t'))\rangle)\\ &= \sigma \circ \Psi((\langle e, t\rangle, \langle e', t'\rangle)).\end{aligned}$$ □

B. Application to the 20-State Encoder

Lemma 3.1 suggests an approach to constructing a sliding-block-decodable encoder by applying a sequence of label consistent out-splittings. In this section, we demonstrate such an encoder construction for the $(d, k; c) = (1, 3; 3)$ constrained system. The code has rate $4 : 8$ and is $(3, 5)$-sliding-block-decodable. The decoder window therefore has total length 9 codewords, or 72 channel symbols. The encoder has twenty states.

The irreducible Shannon cover G of the $(d, k; c) = (1, 3; 3)$ constrained system is shown in Fig. 10 [3], [4]. States are denoted by $(\boldsymbol{s}, \boldsymbol{t})$, where $\boldsymbol{s}$ is the accumulated charge and $\boldsymbol{t}$ is the zero-runlength.

The graph has period 2, so the second power G^2 decomposes into two irreducible graphs. We will work with the graph H shown in Fig. 11, where the edge labels A, B, C correspond to the 2-bit words 00, 10, and 01, respectively.

TABLE I
FOURTH-POWER OF H (TABULAR FORM)

	1	2	3	4	5	6	7
1	$CCCC$	$CABC$	$CCCA$	$CACC$	$CACA$	$CABB$	
2	$ABCC$	$CCCC$ $ACAC$	$ABCA$	$CCAC$ $ACCC$ $ABBC$	$CCAB$ $ACCA$ $ABBB$	$ACAB$	$ABBA$
3	$CACC$	$BCCC$ $BBBC$ $CCAC$	$CACA$	$BCAC$ $BBCC$ $CCCC$ $CABC$	$BCAB$ $BBCA$ $CCCA$ $CABB$	$BBBB$ $CCAB$ $BBAB$	$CABA$
4	$CACC$	$ACCC$ $ABBC$ $CCAC$	$CACA$	$ACAC$ $ABCC$ $CCCC$ $CABC$	$ACAB$ $ABCA$ $CCCA$ $CABB$	$ABBB$ $CCAB$ $ABAB$	$CABA$
5	$CCCC$ $BBCC$	$CABC$ $BCAC$	$CCCA$ $BBCA$	$CACC$ $BCCC$ $BBBC$ $BABC$	$CACA$ $BCCA$ $BBBB$ $BABB$ $CCCA$	$CABB$ $BCAB$	$BBBA$ $BABA$
6	$CACC$	$ABBC$ $BBBC$ $BCCC$ $CCAC$	$CACA$	$ABCC$ $BBCC$ $CCCC$ $CABC$ $BCAC$	$ABCA$ $CABB$ $BBCA$ $BCAB$	$ABBB$ $ABAB$ $BBBB$ $BBAB$ $CCAB$	$CABA$
7	$BBCC$	$BCAC$	$BBCA$	$BCCC$ $BBBC$ $BABC$	$BCCA$ $BBBB$ $BABB$	$BCAB$	$BBBA$ $BABA$

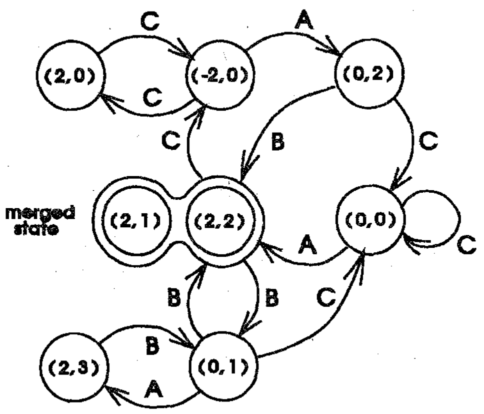

Fig. 11. The graph H, one of two components of the second power of the Shannon cover.

The graph is obtained from one of the components of G^2 by merging the states corresponding to accumulated charge value 2 and zero-runlength values 1 and 2, and it is the irreducible Shannon cover of the constraint it presents. The capacity of the system is exactly 1, and $\boldsymbol{v}^T = [1\,2\,3\,3\,3\,4\,2]$ is a positive integer eigenvector for the eigenvalue $n = 2$. Of all such positive eigenvectors, it minimizes the maximum component. The sequence BA is a homing word for state 7; that is, any path of length 2 that generates this sequence must terminate in state 7. Moreover, any sequence generated by a path passing through state 7 must contain BA. From properties of the Shannon cover [7], it follows that any such sequence must, therefore, be generated by a unique path in H. We refer to a state having this property as a *good* state. Note that state 7 is in fact the only good state in H. An edge that terminates in a good state is called a *good* edge. Therefore, in H, the only good edge is the unique edge entering state 7, labeled A. States and edges that are not good are called *bad*.

The code rate will be 4 : 8, so the starting point for the construction will be the graph H^4, the fourth power of H. Table I describes H^4. Note that the state 7 is the only good state in H^4 and the good edges are those which end at state 7.

One of the steps in the proof of the main theorem in [7] involves performing state-splitting operations to generate many good edges. In order to do this, a scaled eigenvector with components divisible by n^2 is used to guide the splitting. This generates many additional states. In this construction, we want to avoid the creation of a large number of states, so we use a different approach to generate some good edges and good states.

The next two steps involve in-splittings that will enable the creation of additional good states in subsequent out-splittings.

Step 1: In-split state 4, using the partition of incoming edges $IE_4 = IE_4^a \cup IE_4^b$ determined by

$$IE_4^a = \left\{3 \xrightarrow{BBCC} 4, 6 \xrightarrow{BBCC} 4, 3 \xrightarrow{CCCC} 4, 6 \xrightarrow{CCCC} 4\right\}.$$

Denote the descendant states by $4a$ and $4b$.

Step 2: In-split state $4b$, using the partition $IE_{4b} = IE_{4b}^b \cup IE_{4b}^c$ determined by

$$IE_{4b}^b = \left\{4b \xrightarrow{CABC} 4b\right\}.$$

The three states created by this sequence of in-splittings are denoted $4A$, $4B$, $4C$.

The next five steps are rounds of independent out-splittings that satisfy the consistency conditions described in Section III-A. The specific splittings are given in detail below. As in [1], we will use superscripts when we wish to identify descendant states. Edges will be denoted by their destination state t and their label w in the form t/w.

Step 3: In this round, we out-split states $4A$, 6, and 7. We also combine the weight-3 descendant of state 6 with state 3 to simplify the construction. (We remark that an equivalent out-splitting of state 6 and merging of one weight-3 descendant with state 3 could have been carried out directly on the graph H in Fig. 11, but we prefer to present the construction using H^4 as the starting graph.)

a) Out-split state $4A^{1,3}$ into states $4A^1$ and $4A^{2,3}$, according to the partition

$$E^1_{4A} = \left\{ \frac{1}{CACC}, \frac{3}{CACA}, \frac{4C}{CABC}, \frac{4C}{CCCC}, \frac{5}{CABB}, \frac{5}{CCCA} \right\}$$

$$E^{2,3}_{4A} = \left\{ \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{2}{CCAC}, \frac{3}{ABAB}, \frac{3}{ABBB}, \frac{3}{CCAB}, \frac{4C}{ABCC}, \frac{4C}{ACAC}, \frac{5}{ABCA}, \frac{5}{ACAB}, \frac{6}{ABAB}, \frac{6}{ABBB}, \frac{6}{CCAB}, \frac{7}{CABA} \right\}.$$

Note that state $4A^{2,3}$ is a good state.

b) Out-split state 6 into 6^1 and $6^{2,4}$.

$$E^1_6 = \left\{ \frac{2}{ABBC}, \frac{4C}{ABCC}, \frac{5}{ABCA}, \frac{6}{ABBB}, \frac{6}{ACAB} \right\}$$

$$E^{2,4}_6 = \left\{ \frac{1}{CACC}, \frac{2}{BBBC}, \frac{2}{BCCC}, \frac{2}{CCAC}, \frac{3}{CACA}, \frac{4C}{BBCC}, \frac{4C}{BBCC}, \frac{4C}{CCCC}, \frac{4C}{CABC}, \frac{4C}{BCAC}, \frac{5}{CCCA}, \frac{5}{CABB}, \frac{5}{BBCA}, \frac{5}{BCAB}, \frac{6}{BBBB}, \frac{6}{CCAB}, \frac{6}{BBAB}, \frac{7}{CABA} \right\}.$$

State $6^{2,4}$ can be merged with state 3. We will henceforth denote 6^1 by 6, and this is, in fact, a good state.

c) Out-split state 7 into 7^1 and 7^2 according to the partition

$$E^1_7 = \left\{ \frac{1}{BBCC}, \frac{2}{BCAC}, \frac{2}{BCAC}, \frac{3}{BBCA}, \frac{3}{BCAB}, \frac{4C}{BABC}, \frac{4C}{BBBC}, \frac{6}{BCAB} \right\}$$

$$E^2_7 = \left\{ \frac{4C}{BCCC}, \frac{5}{BABB}, \frac{5}{BBBB}, \frac{5}{BCCA}, \frac{7}{BABA}, \frac{7}{BBBA} \right\}.$$

Note that states 7^1 and 7^2 are both good states.

Step 4: Out-split state $3^{1,3}$ into states 3^1 and $3^{2,3}$, according to the partition

$$E^1_3 = \left\{ \frac{2}{CCAC}, \frac{3}{BBAB}, \frac{3}{CCAB}, \frac{4A^{2,3}}{BBCC}, \frac{4A^{2,3}}{CCCC}, \frac{6}{BBBB}, \frac{6}{CCAB}, \frac{7}{CABA} \right\}$$

$$E^{2,3}_3 = \left\{ \frac{1}{CACC}, \frac{2}{BBBC}, \frac{2}{BCCC}, \frac{3}{BBBB}, \frac{3}{CACA}, \frac{4A^1}{BBCC}, \frac{4A^1}{CCCC}, \frac{4C}{BCAC}, \frac{4C}{CABC}, \frac{5}{BBCA}, \frac{5}{BCAB}, \frac{5}{CABB}, \frac{5}{CCCA}, \frac{6}{BBAB} \right\}.$$

Note that state 3^1 is a good state.

Step 5: In this round, we out-split states $4B$ and 5.

1) Out-split state $4B^{1,3}$ into states $4B^1$ and $4B^{2,3}$, according to the partition

$$E^1_{4B} = \left\{ \frac{1}{CACC}, \frac{3^1}{ABAB}, \frac{3^1}{CACA}, \frac{3^{2,3}}{CACA}, \frac{4B}{CABC}, \frac{4B}{CABC}, \frac{5}{CABB}, \frac{5}{CCCA}, \frac{7}{CABA} \right\}$$

$$E^{2,3}_{4B} = \left\{ \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{2}{CCAC}, \frac{3^{2,3}}{ABAB}, \frac{3^1}{ABBB}, \frac{3^{2,3}}{ABBB}, \frac{3^1}{CCAB}, \frac{3^{2,3}}{CCAB}, \frac{4C}{ABCC}, \frac{4C}{ACAC}, \frac{4C}{CCCC}, \frac{5}{ABCA}, \frac{5}{ACAB}, \frac{6}{ABAB}, \frac{6}{ABBB}, \frac{6}{CCAB} \right\}.$$

Note that state $4B^1$ is a good state.

2) Out-split state $5^{1,3}$ into states 5^1 and $5^{2,3}$, according to the partition

$$E^1_5 = \left\{ \frac{3^1}{BBCA}, \frac{3^1}{BCAB}, \frac{3^1}{CABB}, \frac{3^1}{CCCA}, \frac{5}{BABB}, \frac{5}{BCCA}, \frac{6}{BCAB}, \frac{6}{CABB}, \frac{7}{BABA}, \frac{7}{BBBA} \right\}$$

$$E^{2,3}_5 = \left\{ \frac{1}{BBCC}, \frac{1}{CCCC}, \frac{2}{BCAC}, \frac{2}{CABC}, \frac{3^{2,3}}{BBCA}, \frac{3^{2,3}}{BCAB}, \frac{3^{2,3}}{CABB}, \frac{3^{2,3}}{CCCA}, \frac{4C}{BABC}, \frac{4C}{BBBC}, \frac{4C}{BCCC}, \frac{4C}{CACC}, \frac{5}{BBBB}, \frac{5}{CACA} \right\}.$$

Note that state 5^1 is a good state.

Step 6: In this round, we out-split state $4C^{1,3}$ into states $4C^1$ and $4C^{2,3}$, according to the partition

$$E^1_{4C} = \left\{ \frac{3^1}{ABAB}, \frac{3^1}{ABBB}, \frac{3^1}{CACA}, \frac{3^1}{CCAB}, \frac{3^{2,3}}{ABAB}, \frac{4B^1}{CABC}, \frac{5^1}{ABCA}, \frac{5^1}{ACAB}, \frac{5^1}{CABB}, \frac{5^1}{CCCA}, \frac{6}{ABAB}, \frac{6}{ABBB}, \frac{6}{CCAB}, \frac{7}{CABA} \right\}$$

$$E^{2,3}_{4C} = \left\{ \frac{1}{CACC}, \frac{2}{CCAC}, \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{3^{2,3}}{CACA}, \frac{3^{2,3}}{CCAB}, \frac{3^{2,3}}{ABBB}, \frac{4B^{2,3}}{CABC}, \frac{4C}{ABCC}, \frac{4C}{CCCC}, \frac{4C}{ACAC}, \frac{5^{2,3}}{CABB}, \frac{5^{2,3}}{CCCA}, \frac{5^{2,3}}{ABCA}, \frac{5^{2,3}}{ACAB} \right\}.$$

Note that state $4C^1$ is a good state.

Step 7: In this round, we out-split states 2, $3^{2,3}$, $4A^{2,3}$, $4B^{2,3}$, $4C^{2,3}$, and $5^{2,3}$.

a) Out-split state 2 into states 2^1 and 2^2, according to the partition

$$E^1_2 = \left\{ \frac{2}{CCCC}, \frac{3^1}{ABCA}, \frac{3^1}{ACAB}, \frac{4C^1}{ABBC}, \frac{4C^1}{ACCC}, \frac{4C^1}{CCAC}, \frac{4C^{2,3}}{CCAC}, \frac{5^1}{ACCA}, \frac{5^1}{CCAB}, \frac{5^{2,3}}{CCAB}, \frac{6}{ACAB}, \frac{7}{ABBA} \right\}$$

$$E^2_2 = \left\{ \frac{1}{ABCC}, \frac{2}{ACAC}, \frac{3^{2,3}}{ABCA}, \frac{3^{2,3}}{ACAB}, \frac{4C^{2,3}}{ABBC}, \frac{4C^{2,3}}{ACCC}, \frac{5^1}{ABBB}, \frac{5^{2,3}}{ABBB}, \frac{5^{2,3}}{ACCA} \right\}.$$

b) Out-split state $3^{2,3}$ into states 3^2 and 3^3, according to the partition

$$E^2_3 = \left\{ \frac{3^{2,3}}{BBBB}, \frac{3^{2,3}}{CACA}, \frac{4C^{2,3}}{BCAC}, \frac{4C^{2,3}}{CABC}, \frac{5^{2,3}}{BBCA}, \frac{5^{2,3}}{BCAB}, \frac{5^{2,3}}{CABB}, \frac{5^{2,3}}{CCCA} \right\}$$

$$E^3_3 = \left\{ \frac{1}{CACC}, \frac{2}{BBBC}, \frac{2}{BCCC}, \frac{3^1}{BBBB}, \frac{3^1}{CACA}, \frac{4A^1}{BBCC}, \frac{4A^1}{CCCC}, \frac{4C^1}{BCAC}, \frac{4C^1}{CABC}, \frac{5^1}{BBCA}, \frac{5^1}{BCAB}, \frac{5^1}{CABB}, \frac{5^1}{CCCA}, \frac{6}{BBAB} \right\}.$$

c) Out-split state $4A^{2,3}$ into states $4A^2$ and $4A^3$, according to the partition

$$E^2_{4A} = \left\{ \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{3^{2,3}}{ABAB}, \frac{3^{2,3}}{ABBB}, \frac{4C^{2,3}}{ABCC}, \frac{4C^{2,3}}{ACAC}, \frac{5^{2,3}}{ABCA}, \frac{5^{2,3}}{ACAB} \right\}$$

$$E^3_{4A} = \left\{ \frac{2}{CCAC}, \frac{3^1}{ABAB}, \frac{3^1}{ABBB}, \frac{3^1}{CCAB}, \frac{3^{2,3}}{CCAB}, \frac{4C^1}{ABCC}, \frac{4C^1}{ACAC}, \frac{5^1}{ABCA}, \frac{5^1}{ACAB}, \frac{6}{ABAB}, \frac{6}{ABBB}, \frac{6}{CCAB}, \frac{7}{CABA} \right\}.$$

d) Out-split state $4B^{2,3}$ into states $4B^2$ and $4B^3$, according to the partition

$$E^2_{4B} = \left\{ \frac{2}{CCAC}, \frac{3^1}{ABBB}, \frac{3^1}{CCAB}, \frac{3^{2,3}}{CCAB}, \frac{4C^1}{ABCC}, \frac{4C^1}{ACAC}, \frac{4C^1}{CCCC}, \frac{4C^{2,3}}{CCCC}, \frac{5^1}{ABCA}, \frac{5^1}{ACAB}, \frac{6}{ABAB}, \frac{6}{ABBB}, \frac{6}{CCAB} \right\}$$

$$E^3_{4B} = \left\{ \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{3^{2,3}}{ABAB}, \frac{3^{2,3}}{ABBB}, \frac{4C^{2,3}}{ABCC}, \frac{4C^{2,3}}{ACAC}, \frac{5^{2,3}}{ABCA}, \frac{5^{2,3}}{ACAB} \right\}.$$

e) Out-split state $4C^{2,3}$ into states $4C^2$ and $4C^3$, according to the partition

$$E^2_{4C} = \left\{ \frac{1}{CACC}, \frac{2}{CCAC}, \frac{3^{2,3}}{CACA}, \frac{3^{2,3}}{CCAB}, \frac{4B^{2,3}}{CABC}, \frac{4C^1}{ABCC}, \frac{4C^{2,3}}{CCCC}, \frac{5^{2,3}}{CABB}, \frac{5^{2,3}}{CCCA} \right\}$$

$$E^3_{4C} = \left\{ \frac{2}{ABBC}, \frac{2}{ACCC}, \frac{3^{2,3}}{ABBB}, \frac{4C^1}{ACAC}, \frac{4C^1}{CCCC}, \frac{4C^{2,3}}{ABCC}, \frac{4C^{2,3}}{ACAC}, \frac{5^{2,3}}{ABCA}, \frac{5^{2,3}}{ACAB} \right\}.$$

f) Out-split state $5^{2,3}$ into states 5^2 and 5^3, according to the partition

$$E^2_{5^{2,3}} = \left\{ \frac{2}{BCAC}, \frac{2}{CABC}, \frac{3^{2,3}}{BBCA}, \frac{3^{2,3}}{BCAB}, \frac{3^{2,3}}{CABB}, \frac{3^{2,3}}{CCCA}, \frac{5^{2,3}}{BBBB}, \frac{5^{2,3}}{CACA} \right\}$$

$$E^3_{5^{2,3}} = \left\{ \frac{1}{BBCC}, \frac{1}{CCCC}, \frac{4C^1}{BABC}, \frac{4C^1}{BBBC}, \frac{4C^1}{BCCC}, \frac{4C^1}{CACC}, \frac{4C^{2,3}}{BABC}, \frac{4C^{2,3}}{BBBC}, \frac{4C^{2,3}}{BCCC}, \frac{4C^{2,3}}{CACC}, \frac{5^1}{BBBB}, \frac{5^1}{CACA} \right\}.$$

TABLE II
SUMMARY OF 20-STATE ENCODER CONSTRUCTION

Step	1	2	3	Merge	4	5	6	7	Merge	Final
1										1
$2^{1,2}$								2^1		2
								2^2		3
$3^{1,3}$				$3^{1,3}$	3^1					4
					$3^{2,3}$			3^2		5
								3^3		6
$4^{1,3}$	$4a^{1,3}$	$4A^{1,3}$	$4A^1$							7
			$4A^{2,3}$					$4A^2$	$4A^2$	8
								$4A^3$		9
	$4b^{1,3}$	$4B^{1,3}$				$4B^1$				10
						$4B^{2,3}$		$4B^2$	$4B^2$	
								$4B^3$		11
		$4C^{1,3}$					$4C^1$			12
							$4C^{2,3}$	$4C^2$		13
								$4C^3$		14
$5^{1,3}$						5^1				15
						$5^{2,3}$		5^2		16
								5^3		17
$6^{1,4}$			6^1							18
			$6^{2,4}$	$6^{2,4}$						
$7^{1,2}$			7^1							19
			7^2							20

Step 8: The states $4A^2$ and $4B^2$ have the same outgoing picture, and we therefore merge them.

This concludes the construction of the encoder graph. Table II summarizes the construction, and indicates the correspondence between the states in the graph and the states numbered 1–20 in the encoder graph.

C. Consistent Input Tag Assignment

We now describe in detail the input tag assignment that produces a sliding-block-decodable encoder. The good states in the encoder graph are states 4, 9, 10, 12, 15, 18, 19, 20 of Table II, corresponding to states 3^1, $4A^3$, $4B^1$, $4C^1$, 5^1, 6, 7^1, 7^2. The remaining states are bad, and some of the edges outgoing from each such state will have to be labeled consistently with edges outgoing from some other state.

In fact, one can verify that the round of out-splitting performed in Step (i), where $3 \leq i \leq 7$, is a $(3, i-3)$-label-consistent out-splitting. For $2 \leq i \leq 7$, denote by $G^{(i)}$ the graph that results after performing Steps 1) through i). Set $H^{(i)} = \boldsymbol{H}(G^{(i)}; 3, i-2)$ for $2 \leq i \leq 7$. The value $m = 3$ is chosen because looking back any further does not eliminate any more pairs (s, s') (states) from $G^{(2)} * G^{(2)}$ than those that are already eliminated in passing to the subgraph $\boldsymbol{H}(G^{(2)}; 3, 0)$ of $G^{(2)} * G^{(2)}$. It follows that $H^{(i)}$ can be computed along with $G^{(i)}$ as in Lemma 3.1 while performing Steps 3)–7). The consistency problem involves exactly those sets of edges of $G^{(7)}$ related by the connectedness equivalence relation (defined in Section II) generated by the pairs (e, e') of $(3, 5)$-adjacent edges of $G^{(7)}$ that occur as edges of $H^{(7)}$. If we manage to find an input tagging that colors all the edges in the equivalence class $[e]$ the same (for each equivalence class) then the resulting encoder will be at worst $(3, 5)$-sliding-block-decodable.

On the other hand, to construct our 20-state, rate 4:8, $(3, 5)$-sliding-block-decodable $(1, 3; 3)$ encoder, we performed five rounds of out-splitting (in Steps 3)–7)). In each of the last four of these five rounds, at least one state was split according to a partition of its outgoing edges into two sets, E and E', where the set E contains an edge e leading to one of two descendants s of a state t that was out-split in the previous round, and the set E' contains an edge e' having the same label as e but leading to a distinct descendant s' of the state t. It follows from this that in the resulting graph there are two paths $e_0e_1e_2e_3e_4$ and $e'_0e'_1e'_2e'_3e'_4$ emanating from the same state, with distinct first edges $e_0 \neq e'_0$, but that generate the same label sequence $L(e_0e_1e_2e_3e_4) = L(e'_0e'_1e'_2e'_3e'_4)$. It follows from this observation alone that (for any input tag assignment $\mathcal{I}$) any encoder based on the resulting graph is at best $(m, 5)$-sliding-block-decodable for some $m \geq 0$. (One must sometimes look ahead at least five symbols to decide whether to decode to $\mathcal{I}(e_0)$ or to $\mathcal{I}(e'_0)$.) Thus anticipation $a = 5$ is also the best that can be hoped for given that five rounds of out-splitting were performed to produce the encoder [12].

The consistency problem involves the following subsets of states: $\{1, 2, 7, 11, 13\}$, $\{3, 8, 14\}$, $\{5, 16\}$, and $\{6, 17\}$.

Tables III–V confirm that a consistent input tag assignment can be made for the encoder graph generated by the sequence of out-splittings and mergers. For each of the subsets of bad

TABLE III
CONSISTENT SPLITTING CHECK FOR BAD STATES $\{1, 2, 7, 11, 13\}$

1	2	7	11	13
1 / $CCCC$	2/ $CCCC$	13/ $CCCC$	13 / $CCCC$	13/ $CCCC$
2/$CABC$		13/$CABC$		11/$CABC$
3/$CABC$		14/$CABC$		8/$CABC$
5/$CABB$		16/$CABB$		16/$CABB$
5/$CCCA$		16/$CCCA$		16/$CCCA$
6/$CABB$		17/$CABB$		17/$CABB$
6/$CCCA$		17/$CCCA$		17/$CCCA$
13/$CACC$		1/$CACC$		1/$CACC$
16/$CACA$		5/$CACA$		5/$CACA$
17/$CACA$		6/$CACA$		6/$CACA$
	3/$CCCC$	14/$CCCC$	14/$CCCC$	14/$CCCC$
	13/$CCAC$		2/$CCAC$	2/$CCAC$
	14/$CCAC$		3/$CCAC$	3/$CCAC$
	16/$CCAB$		5/$CCAC$	5/$CCAB$
	17/$CCAB$		6/$CCAB$	6/$CCAB$

TABLE IV
CONSISTENT SPLITTING CHECK FOR BAD STATES $\{3, 8, 14\}$

3	8	14
1/$ABCC$	13/ $ABCC$	13/ $ABCC$
2/$ACAC$	13/$ACAC$	13/$ACAC$
3/$ACAC$	14/$ACAC$	14/$ACAC$
5/$ABCA$	16/$ABCA$	16/$ABCA$
5/$ACAB$	16/$ACAB$	16/$ACAB$
6/$ABCA$	17/$ABCA$	17/$ABCA$
6/$ACAB$	17/$ACAB$	17/$ACAB$
13/$ABBC$	2/$ABBC$	2/$ABBC$
13/$ACCC$	2/$ACCC$	2/$ACCC$
14/$ABBC$	3/$ABBC$	3/$ABBC$
14/$ACCC$	3/$ACCC$	3/$ACCC$
16/$ABBB$	5/$ABBB$	5/$ABBB$
17/$ABBB$	6/$ABBB$	6/$ABBB$

TABLE V
CONSISTENT SPLITTING CHECK FOR BAD STATES $\{5, 16\}$ AND $\{6, 17\}$

5	16	6	17
5/$BBBB$	16/ $BBBB$	1/ $CACC$	13/ $CACC$
5/$CACA$	16/$CACA$	2/$BBBC$	13/$BBBC$
6/$BBBB$	17/$BBBB$	2/$BCCC$	13/$BCCC$
6/$CACA$	17/$CACA$	3/$BBBC$	14/$BBBC$
13/$BCAC$	2/$BCAC$	3/$BCCC$	14/$BCCC$
13/$CABC$	2/$CABC$	7/$BBCC$	1/$BBCC$
14/$BCAC$	3/$BCAC$	7/$CCCC$	1/$CCCC$
14/$CABC$	3/$CABC$		
16/$BBCA$	5/$BBCA$		
16/$BCAB$	5/$BCAB$		
16/$CABB$	5/$CABB$		
16/$CCCA$	5/$CCCA$		
17/$BBCA$	6/$BBCA$		
17/$BCAB$	6/$BCAB$		
17/$CABB$	6/$CABB$		
17/$CCCA$	6/$CCCA$		

TABLE VI
COMPLETE SPECIFICATION OF 20-STATE $(1,3;3)$ ENCODER

State / Data	0000	0001	0010	0011
1	1 / $CCCC$	2/ $CABC$	3/ $CABC$	5 / $CABB$
2	2/$CCCC$	4/$ABCA$	4/$ACAB$	12/$ABBC$
3	1/$ABCC$	2/$ACAC$	3/$ACAC$	5/$ABCA$
4	2/$CCAC$	3/$CCAC$	4/$BBAB$	4/$CCAB$
5	5/$BBBB$	5/$CACA$	6/$BBBB$	6/$CACA$
6	1/$CACC$	2/$BBBC$	2/$BCCC$	3/$BBBC$
7	13/$CCCC$	13/$CABC$	14/$CABC$	16/$CABB$
8	13/$ABCC$	13/$ACAC$	14/$ACAC$	16/$ABCA$
9	2/$CCAC$	3/$CCAC$	4/$ABAB$	4/$ABBB$
10	1/$CACC$	4/$ABAB$	4/$CACA$	5/$CACA$
11	13/$CCCC$	4/$ABBB$	4/$CCAB$	12/$ABCC$
12	4/$ABAB$	4/$ABBB$	4/$CACA$	4/$CCAB$
13	13/$CCCC$	11/$CABC$	8/$CABC$	16/$CABB$
14	13/$ABCC$	13/$ACAC$	14/$ACAC$	16/$ABCA$
15	4/$BBCA$	4/$BCAB$	4/$CABB$	4/$CCCA$
16	16/$BBBB$	16/$CACA$	17/$BBBB$	17/$CACA$
17	13/$CACC$	13/$BBBC$	13/$BCCC$	14/$BBBC$
18	2/$ABBC$	3/$ABBC$	4/$ABAB$	4/$ABBB$
19	1/$BBCC$	2/$BCAC$	3/$BCAC$	4/$BBCA$
20	12/$BCCC$	13/$BCCC$	14/$BCCC$	15/$BABB$

states, the bad edges that must be assigned the same tag are aligned in rows. As can be seen, no edge is forced by the consistency requirement to appear in more than one row of the table, ensuring that consistent tagging can be achieved. Applying the testing algorithm described in Section I-C with $0 \le m \le 3$, one can confirm that the smallest m for which the tagged encoder is $(m, 5)$-sliding-block-decodable is $m = 3$. Therefore, the required decoder window size is $m + a + 1 = 9$ codewords, or 72 code symbols.

The final input tag assignment is reflected in the definition of the tagged encoder $\mathcal{E}$ in Tables VI–IX. The rows in Tables VI–IX correspond to the encoder states, and the columns represent the 16 possible 4-bit inputs. The entries in the table represent the next state t and output codeword c in the usual fashion t/c.

IV. A 104-STATE, RATE 4:8, (0, 2)-SLIDING-BLOCK-DECODABLE $(1, 3; 3)$ ENCODER

In this section, we construct another rate $4 : 8$ encoder into the $(d, k; c) = (1, 3; 3)$ system. The code has a memory $m = 0$ and an anticipation $a = 2$ codewords (of length 8 each). Thus the window has total length $(1+2)\cdot 8 = 24$ code symbols. The code is constructed from a component of the eighth power of the minimal co-deterministic presentation using two rounds of out-splitting. (The minimal co-deterministic presentation is the fourth power of the transpose of the companion component of H in the second power of the $(1, 3; 3)$ Shannon cover. It is represented in tabular form in Table X.) This small window is achieved at the price of having many more encoder states than our previous code has: this encoder has $13 \cdot 8 = 104$ states.

The construction can be cast as an instance of a general procedure that starts with a (not necessarily deterministic) presentation $G^{(0)}$ of a constrained system and with a target out-degree n for an encoder into the constrained system. The general procedure is not guaranteed to produce an encoder, but it does so in the case of the $(d, k; c) = (1, 3; 3)$ constraint. Also, it is easier to describe even our particular $(d, k; c) = (1, 3; 3)$ code construction in this more general context.

A. A Matrix Generalization of the State-Splitting Algorithm

We start with a labeled graph $G^{(0)}$ presenting a constrained system. We assume that $G^{(0)}$ has no parallel edges with the

TABLE VII
COMPLETE SPECIFICATION OF 20-STATE $(1,3;3)$ ENCODER (*cont.*)

State / Data	0100	0101	0110	0111
1	$5/\ CCCA$	$6/CABB$	$6/CCCA$	$13/\ CACC$
2	$12/ACCC$	$12/CCAC$	$15/ACCA$	$15/CCAB$
3	$5/ACAB$	$6/ABCA$	$6/ACAB$	$13/ABBC$
4	$5/BBAB$	$5/CCAB$	$6/BBAB$	$6/CCAB$
5	$13/BCAC$	$13/CABC$	$14/BCAC$	$14/CABC$
6	$3/BCCC$	$7/BBCC$	$7/CCCC$	$4/BBBB$
7	$16/CCCA$	$17/CABB$	$17/CCCA$	$1/CACC$
8	$16/ACAB$	$17/ABCA$	$17/ACAB$	$2/ABBC$
9	$4/CCAB$	$5/CCAB$	$6/CCAB$	$12/ABCC$
10	$6/CACA$	$8/CABC$	$10/CABC$	$11/CABC$
11	$12/ACAC$	$12/CCCC$	$15/ABCA$	$15/ACAB$
12	$5/ABAB$	$6/ABAB$	$10/CABC$	$15/ABCA$
13	$16/CCCA$	$17/CABB$	$17/CCCA$	$1/CACC$
14	$16/ACAB$	$17/ABCA$	$17/ACAB$	$2/ABBC$
15	$15/BABB$	$15/BCCA$	$16/BABB$	$16/BCCA$
16	$2/BCAC$	$2/CABC$	$3/BCAC$	$3/CABC$
17	$14/BCCC$	$1/BBCC$	$1/CCCC$	$12/BABC$
18	$5/ABAB$	$5/ABBB$	$6/ABAB$	$6/ABBB$
19	$4/BCAB$	$5/BBCA$	$5/BCAB$	$6/BBCA$
20	$15/BBBB$	$15/BCCA$	$16/BABB$	$16/BBBB$

TABLE VIII
COMPLETE SPECIFICATION OF 20-STATE $(1,3;3)$ ENCODER (*cont.*)

State / Data	1000	1001	1010	1011
1	$14/\ CACC$	$16/\ CACA$	$17/\ CACA$	$4\ /\ CABB$
2	$18/ACAB$	$19/ABBA$	$20/ABBA$	$3/CCCC$
3	$13/ACCC$	$14/ABBC$	$14/ACCC$	$16/ABBB$
4	$8/BBCC$	$8/CCCC$	$9/BBCC$	$9/CCCC$
5	$16/BBCA$	$16/BCAB$	$16/CABB$	$16/CCCA$
6	$4/CACA$	$12/BCAC$	$12/CABC$	$15/BBCA$
7	$4/CACA$	$5/CACA$	$6/CACA$	$14/CCCC$
8	$2/ACCC$	$3/ABBC$	$3/ACCC$	$5/ABBB$
9	$12/ACAC$	$15/ABCA$	$15/ACAB$	$18/ABAB$
10	$15/CABB$	$15/CCCA$	$16/CABB$	$16/CCCA$
11	$18/ABAB$	$18/ABBB$	$18/CCAB$	$14/CCCC$
12	$15/ACAB$	$15/CABB$	$15/CCCA$	$18/ABAB$
13	$12/ABCC$	$5/CACA$	$6/CACA$	$14/CCCC$
14	$2/ACCC$	$3/ABBC$	$3/ACCC$	$5/ABBB$
15	$17/BABB$	$17/BCCA$	$18/BCAB$	$18/CABB$
16	$5/BBCA$	$5/BCAB$	$5/CABB$	$5/CCCA$
17	$12/BBBC$	$12/BCCC$	$12/CACC$	$13/BABC$
18	$12/ABCC$	$13/ABCC$	$14/ABCC$	$15/ABCA$
19	$6/BCAB$	$12/BABC$	$12/BBBC$	$13/BABC$
20	$16/BCCA$	$17/BABB$	$17/BBBB$	$17/BCCA$

same label (if it does, delete all but one, and it still presents the same constrained system). We are also given a target out-degree n for an encoder into the constrained system. For this target n to be feasible, the capacity of the constrained system cannot fall below $\log(n)$. This implies in particular that the largest eigenvalue of the transition matrix T of $G^{(0)}$ cannot fall below $\log(n)$; if the presentation is not lossless, all the more so!

We construct a new object $H^{(0)}$ as follows. We cannot resist the temptation to call this new object a *polygraph*, and we say $G^{(0)}$ *underlies* $H^{(0)}$. The polygraph $H^{(0)}$ has a single state s_0 corresponding to an ordered list of the states of $G^{(0)}$. For each distinct symbol that occurs as an edge label in $G^{(0)}$, there corresponds a self-loop edge e in $H^{(0)}$ from s_0 to itself. At this point, $H^{(0)}$ seems far too simple to be of any use, and it is. Now here is where the complexity of $G^{(0)}$ is reflected in $H^{(0)}$: The self-loop e in $H^{(0)}$ corresponding to edge label w is labeled not only by w, but by an integer matrix $M^{(w)}$

TABLE IX
COMPLETE SPECIFICATION OF 20-STATE $(1,3;3)$ ENCODER (*cont.*)

State / Data	1100	1101	1110	1111
1	$4/\ CCCA$	$12/\ CACC$	$15/\ CACA$	$18/\ CABB$
2	$13/CCAC$	$14/CCAC$	$16/CCAB$	$7/CCAB$
3	$16/ACCA$	$17/ABBB$	$17/ACCA$	$15/ABBB$
4	$18/BBBB$	$18/CCAB$	$19/CABA$	$20/CABA$
5	$17/BBCA$	$17/BCAB$	$17/CABB$	$17/CCCA$
6	$15/BCAB$	$15/CABB$	$15/CCCA$	$18/BBAB$
7	$12/CCCC$	$12/CABC$	$15/CABB$	$15/CCCA$
8	$5/ABAB$	$6/ABBB$	$6/ABAB$	$14/ABCC$
9	$18/ABBB$	$18/ACAB$	$19/CABA$	$20/CABA$
10	$17/CABB$	$17/CCCA$	$19/CABA$	$20/CABA$
11	$2/CCAC$	$3/CCAC$	$5/CCAB$	$6/CCAB$
12	$18/ABBB$	$18/CCAB$	$19/CABA$	$20/CABA$
13	$2/CCAC$	$3/CCAC$	$5/CCAB$	$6/CCAB$
14	$12/ACAC$	$6/ABBB$	$12/CCCC$	$14/ABCC$
15	$19/BABA$	$19/BBBA$	$20/BABA$	$20/BBBA$
16	$6/BBCA$	$6/BCAB$	$6/CABB$	$6/CCCA$
17	$14/BABC$	$14/CACC$	$15/BBBB$	$15/CACA$
18	$16/ABCA$	$17/ABCA$	$18/ABAB$	$18/ABBB$
19	$13/BBBC$	$14/BABC$	$14/BBBC$	$18/BCAB$
20	$19/BABA$	$19/BBBA$	$20/BABA$	$20/BBBA$

TABLE X
THE SPLITTING OF s_0 IN THE FIRST ROUND

$w(s_0) = (16,8,16,24,8,24,8)$		
out-going edges		
label	**matrix**	**$w(e)$**
	s_1	$(8,0,0,10,0,0,0)$
BABA	$[(1 \to 1),(4 \to 4)]$	$(16,0,0,24,0,0,0)$
BCCC	$[(1 \to 2,3),(4 \to 5,6)]$	$(24,0,0,32,0,0,0)$
BBBC	$[(1 \to 2,3),(4 \to 5,6)]$	$(24,0,0,32,0,0,0)$
BABC	$[(1 \to 5,6),(4 \to 2,3)]$	$(32,0,0,24,0,0,0)$
BCAB	$[(1 \to 6,7),(4 \to 2,3,4)]$	$(32,0,0,48,0,0,0)$
	s_2	$(4,0,0,2,0,0,0)$
BBBA	$[(1 \to 4),(4 \to 1)]$	$(24,0,0,16,0,0,0)$
BCCA	$[(1 \to 3,4),(4 \to 1)]$	$(40,0,0,16,0,0,0)$
	s_3	$(4,0,0,4,0,0,2)$
BABB	$[(1 \to 2,3,4),(4 \to 1)]$	$(48,0,0,16,0,0,0)$
BBBB	$[(1 \to 1),(4 \to 2,3,4),(7 \to 6,7)]$	$(16,0,0,48,0,0,32)$
	s_4	$(0,8,0,0,0,10,0)$
ABCA	$[(2 \to 1),(6 \to 3,4)]$	$(0,16,0,0,0,40,0)$
ABCC	$[(2 \to 5,6),(6 \to 2,3)]$	$(0,32,0,0,0,24,0)$
ABBB	$[(2 \to 6,7),(6 \to 2,3,4)]$	$(0,32,0,0,0,48,0)$
ABAB	$[(2 \to 2,3,4),(6 \to 6,7)]$	$(0,48,0,0,0,32,0)$
ABBA	$[(6 \to 1)]$	$(0,0,0,0,0,16,0)$
	s_5	$(0,0,4,0,0,0,0)$
CACC	$[(3 \to 2,3)]$	$(0,0,24,0,0,0,0)$
CACA	$[(3 \to 3,4)]$	$(0,0,40,0,0,0,0)$
	s_6	$(0,0,6,0,0,4,0)$
CABA	$[(3 \to 1),(6 \to 4)]$	$(0,0,16,0,0,24,0)$
CABC	$[(3 \to 5,6),(6 \to 2,3)]$	$(0,0,32,0,0,24,0)$
CABB	$[(3 \to 2,3,4),(6 \to 1)]$	$(0,0,48,0,0,16,0)$
	s_7	$(0,0,6,0,0,6,0)$
CCCC	$[(3 \to 2,3),(6 \to 5,6)]$	$(0,0,24,0,0,32,0)$
CCAB	$[(3 \to 6,7),(6 \to 2,3,4)]$	$(0,0,32,0,0,48,0)$
CCCA	$[(3 \to 3,4),(6 \to 1)]$	$(0,0,40,0,0,16,0)$
	s_8	$(0,0,0,2,0,0,0)$
BCAC	$[(4 \to 5,6)]$	$(0,0,0,32,0,0,0)$
	s_9	$(0,0,0,6,0,0,6)$
BBCC	$[(4 \to 2,3),(7 \to 5,6)]$	$(0,0,0,24,0,0,32)$
BBAB	$[(4 \to 6,7),(7 \to 2,3,4)]$	$(0,0,0,32,0,0,48)$
BBCA	$[(4 \to 3,4),(7 \to 1)]$	$(0,0,0,40,0,0,16)$
	s_{10}	$(0,0,0,0,4,0,0)$
ACCA	$[(5 \to 1)]$	$(0,0,0,0,16,0,0)$
ACAB	$[(5 \to 2,3,4)]$	$(0,0,0,0,48,0,0)$
	s_{11}	$(0,0,0,0,2,0,0)$
ACAC	$[(5 \to 5,6)]$	$(0,0,0,0,32,0,0)$
	s_{12}	$(0,0,0,0,2,0,0)$
ACCC	$[(5 \to 5,6)]$	$(0,0,0,0,32,0,0)$
	s_{13}	$(0,0,0,0,0,2,0)$
ABBC	$[(6 \to 5,6)]$	$(0,0,0,0,0,32,0)$
	s_{14}	$(0,0,0,0,0,2,0)$
CCAC	$[(6 \to 5,6)]$	$(0,0,0,0,0,32,0)$

indexed by the states of $G^{(0)}$ and defined by

$$M_{ij}^{(w)} = \begin{cases} 1, & \text{if there is an edge } i \xrightarrow{w} j \text{ in } G^{(0)} \\ 0, & \text{otherwise.} \end{cases}$$

Notice that $\sum_w M^{(w)}$ is the ordinary transition matrix T of the graph $G^{(0)}$. We denote the matrix labeling edge e of $H^{(0)}$ by $M(e)$.

Now we assign an integer column vector $\boldsymbol{w}(s_0)$ we call a *weight* to the single state s_0. In fact, we choose $\boldsymbol{w}(s_0)$ to be any approximate right eigenvector $\boldsymbol{r}$ of the ordinary transition matrix T of $G^{(0)}$ corresponding to approximate eigenvalue n.

Now

$$\left(\sum_e M(e)\right)\boldsymbol{w}(s_0) = \left(\sum_w M^{(w)}\right)\boldsymbol{w}(s_0) = T\boldsymbol{w}(s_0) \geq n\boldsymbol{w}(s_0)$$

so at any state u of $H^{(0)}$ (there is only one!) the sum of the *weights* of its following edges (we define the weight $\boldsymbol{w}(e)$ of an edge e to be $M(e)\boldsymbol{w}(\tau(e))$, where $\tau(e)$ is the terminal state of the edge e) is at least n times the weight $\boldsymbol{w}(u)$ of u itself.

A *state-splitting* is defined as usual, except that, for a polygraph, descendant edges inherit the matrix $M(e)$ of their ancestral edges. A state s is split into several states $s_1, \cdots, s_m$, each state corresponding to an atom P_i of a partition of the outgoing edges from state s.

We call a state-splitting of a polygraph H *legal* with respect to a particular weight assignment $\boldsymbol{w}$ to its states if, when state s is split into states $s_1, \cdots, s_m$, using the partition $\{P_1, \cdots, P_m\}$, we have corresponding weights $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m$ satisfying

$$\sum_i \boldsymbol{w}_i = \boldsymbol{w}(s)$$

$$\sum_{e \in P_i} \boldsymbol{w}(e) \geq n\boldsymbol{w}_i, \quad 1 \leq i \leq m.$$

It is easy to verify that if we perform a legal round of state-splitting on a polygraph H, and we assign the weight $\boldsymbol{w}_i$ to the state s_i corresponding to atom P_i, then the resulting polygraph H', like H, satisfies:

> At any state u of H' the sum of the weights of its following edges is at least n times the weight of the state u itself.

We call this the *weight condition* for future reference.

We will perform two legal rounds of state-splitting on the single-state polygraph $H^{(0)}$ constructed starting with a component of the seven-state eighth power of the minimal co-deterministic presentation of the $(d, k; c) = (1, 3; 3)$ constrained system. We will start out with weight assignment $\boldsymbol{w}(s_0) = 8(2, 1, 2, 3, 1, 3, 1)^\top$, 8 times the smallest right eigenvector corresponding to eigenvalue $n = 16$.

We will end up with a polygraph $H^{(2)}$ whose states all have weights that are integer vectors with only 0 or 1 entries. Each state of the polygraph $H^{(2)}$ overlies a list of seven states of an underlying graph $G^{(2)}$, some fictitious (those whose corresponding weight vector entry is 0), and some real (those whose corresponding weight vector entry is 1). Likewise, for each edge e of the polygraph $H^{(2)}$, and for each nonzero $M(e)_{ij}$ there is an edge in $G^{(2)}$ from the ith state underlying the initial state $\sigma(e)$ of e to the jth state underlying the terminal state $\tau(e)$ of e. It follows from the weight condition that each real state of $G^{(2)}$ has at least 16 outgoing edges terminating at real states.

The labeling of a polygraph H' obtained by state-splitting a polygraph H is inherited (as usual in state-splitting) from the labeling of H. The label of any edge of the resulting graph G' underlying H' is defined to be the label of the edge of H' that it underlies.

Suppose $H^{(a)}$ is obtained from the single-state polygraph $H^{(0)}$ by a rounds of legal state-splitting, and that the states of $H^{(a)}$ all have 0–1 weights. Let $G^{(a)}$ be the graph underlying the polygraph $H^{(a)}$. The problem of finding a consistent input tag assignment for $G^{(a)}$ will be reduced to solving a certain integer programming problem with 0–1 constraints. In general, there is no guarantee that it has a solution, but in the particular instance that arises in the construction of our encoder, there is an easy-to-find solution.

Now we formulate this integer programming problem. First, the setup. We fix an integer n, and a labeled graph $G^{(0)}$ having k states. Suppose that $H^{(0)}$ is a polygraph constructed from $G^{(0)}$ as above, having a single state s_0, with a weight $\boldsymbol{w}(s_0)$ so that $H^{(0)}$ satisfies the weight condition for n. Suppose that a polygraph $H^{(a)}$ is obtained from $H^{(0)}$ by a rounds of legal out-splitting, and that the states of $H^{(a)}$ all have 0–1 weight vectors. For each state t of $H^{(a)}$, fix an ordering of its outgoing edges in $H^{(a)}$, say as $e_1, \cdots, e_{d_t}$. Form the k-by-d_t matrix $W(t)$ by setting its ith column to be $\boldsymbol{w}(e_i)$.

Theorem 4.1: Suppose that for each state t of $H^{(a)}$ there is a d_t-by-n matrix $R(t)$ such that:

1) each row of $R(t)$ is an elementary row vector;
2) $W(t)R(t) \geq \boldsymbol{w}(t)[1\,1 \cdots 1]$.

We define an input tag assignment $\mathcal{I}$ of $G^{(a)}$ as follows. Suppose that f is an edge of $G^{(a)}$ from state s to state s'. Let e_i be the edge in $H^{(a)}$ that f underlies, and let t be its initial state in $H^{(a)}$. Define $\mathcal{I}(f) = j$ if and only if $R(t)_{ij} = 1$. Then a subgraph of $G^{(a)}$, equipped with the input tag assignment $\mathcal{I}$, is an encoder. Moreover, this encoder is $(0, a)$-sliding-block-decodable.

Proof: First we show that a subgraph of $G^{(a)}$ is an encoder. We must show that for each input symbol $0 \leq j \leq n-1$, and for each real state s of $G^{(a)}$, there is an edge f with initial state s, with input tag $\mathcal{I}(f) = j$, and with a real terminal state s'. Let t be the state of $H^{(a)}$ that s underlies. Consider the row of the matrix inequality $W(t)R(t) \geq \boldsymbol{w}(t)[1\,1 \cdots 1]$ corresponding to state s (call it the sth row). Since s is a real state underlying t, $\boldsymbol{w}(t)_s = 1$, so the right-hand side is the length-n all-1's row vector $[1\,1 \cdots 1]$. So for each j, $0 \leq j \leq n-1$, there is at least one index i, $1 \leq i \leq d_t$, such that $W(t)_{si} \geq 1$ and $R(t)_{ij} = 1$. Now consider the edge e_i following state t in $H^{(a)}$. Recall $\boldsymbol{w}(e_i) = M(e_i)\boldsymbol{w}(t')$, where t' is the terminal state of e_i in $H^{(a)}$. But $\boldsymbol{w}(e_i)_s = W(t)_{si} \geq 1$, so there is a real state s' underlying state t', for which the column of $M(e_i)$ corresponding to s' has a 1 in the row corresponding to s. Thus there is an edge f in $G^{(a)}$ underlying the edge e_i with initial state s and with terminal state s'. Since $R(t)_{ij} = 1$, we have $\mathcal{I}(f) = j$.

It remains to show that the resulting encoder is sliding-block-decodable with memory 0 and anticipation a. Because $H^{(a)}$ was obtained from $H^{(0)}$ by a rounds of out-splitting, for any sequence $w_0 w_1 \cdots w_a$ of $a+1$ labels, all paths in $H^{(a)}$ labeled by this sequence begin with the same edge, say e. Thus all paths in $G^{(a)}$ labeled by this sequence begin with an edge that underlies e. But the input tag of an edge of $G^{(a)}$ only depends on the edge of $H^{(a)}$ that it underlies. Thus all of paths in $G^{(a)}$ generating the label sequence $w_0 w_1 \cdots w_a$ have initial edges sharing the same input tag. This proves that the encoder is sliding-block-decodable with memory 0 and anticipation a. $\square$

B. Application to the $(1, 3; 3)$ *Constraint*

We start with one of the two components $G^{(0)}$ of the eighth power of the minimal co-deterministic presentation of the $(d, k; c) = (1, 3; 3)$ constrained system. The graph $G^{(0)}$ has seven states. We construct the polygraph $H^{(0)}$ as in the previous subsection. We assign state s_0 of $H^{(0)}$ the weight $\boldsymbol{w}(s_0) = 8 \cdot (2, 1, 2, 3, 1, 3, 1)^\top$, 8 times the smallest right eigenvector of the transition matrix of $G^{(0)}$ corresponding to eigenvalue $n = 16$. We perform the two rounds of out-splitting on $H^{(0)}$, as described below.

1) The First Round of Splitting: There are 32 self-loop edges at state s_0 of the polygraph $H^{(0)}$, one for each of the 32 allowed words of length 8. These edges are listed with their labels, corresponding matrices, and weights in Table X.

Each label is a length 4 word over the symbols $A = 00$, $B = 10$, and $C = 01$. The 7×7 matrix $M^{(w)}$ corresponding to word w is abbreviated as a list of transition specifications, one specification for each nonzero row of $M^{(w)}$. If row i of $M^{(w)}$ has 1's in columns $j_1, \cdots, j_m$, then the transition specification corresponding to row i has the form $(i \rightarrow j_1, \cdots, j_m)$. Incidentally, Table X also completely specifies the transitions of $G^{(0)}$. All the transitions having a given label are listed in the same row of Table X. The weight $M(e)\boldsymbol{w}(\tau(e))$ of each edge is listed in the last column.

The rows of Table X corresponding to edges of $H^{(0)}$ are partitioned into 14 sets. The set P_i of edges of $H^{(0)}$ used to define the state s_i of $H^{(1)}$ in the first round of splitting has rows listed under the header s_i. The weight of $\boldsymbol{w}(s_i)$ of state s_i in $H^{(1)}$ appears in the last column of its header row. One can easily verify that

$$\sum_{e \in P_i} \boldsymbol{w}(e) = 16\boldsymbol{w}(s_i), \qquad 1 \le i \le 14$$

and that

$$\sum_{i=1}^{14} \boldsymbol{w}(s_i) = \boldsymbol{w}(s_0)$$

so the first round of splitting is legal.

2) The Second Round of Splitting: The graph $H^{(1)}$ has 14 states, $s_1, \cdots, s_{14}$, created in the first round of splitting (specified in Table X). To completely specify the second round of splitting, we need to specify 14 different partitions, one partition for each of the 14 states of graph $H^{(1)}$. Thus we need 14 tables, each being similar in size and form to Table X. In fact, if the set of edges in Table X corresponding to state s_i has m_i elements, then the number of outgoing edges from state s_i in $H^{(1)}$ is $14m_i$, because each edge in $H^{(0)}$ has 14 descendants in $H^{(1)}$. For instance, the table corresponding to state s_1 has $5 \cdot 14 = 70$ rows corresponding to edges.

For the sake of brevity, we only present the tables for four states s_3, s_7, s_9, and s_{13} in $H^{(1)}$. The other states have quite similar tables, and these can readily be discovered by the reader, given the example of these four. We choose to present tables for these four states because it will turn out that their descendants in $H^{(2)}$ will account for all of the terminal states of outgoing edges emanating from a particular descendant $s_{3.3}$ in $H^{(2)}$ of state s_3 in $H^{(1)}$. In the next subsection, we plan to use state $s_{3.3}$ of $H^{(2)}$ to illustrate the input tag assignment problem, because it provides the most interesting illustration.

Tables XI–XIV specify the splittings in the second round of states s_3, s_7, s_9, and s_{13} in $H^{(1)}$.

In Tables XI–XIV, the second column lists the terminal state $\tau(e)$ in $H^{(1)}$ of the edge e corresponding to the row. Using this terminal state, and the label listed in the first column, one can readily verify the edge weight $\boldsymbol{w}(e)$ listed in the last column by using Table X to look up $M(e)$ and $\boldsymbol{w}(\tau(e))$, and then calculate $\boldsymbol{w}(e) = M(e)\boldsymbol{w}(\tau(e))$. For instance, the second

TABLE XI
THE SPLITTING OF s_3 IN THE SECOND ROUND

w(s_3) = (4,0,0,4,0,0,2)		
label	$\tau(e)$	w(e)
$s_{3.1}$		(1,0,0,0,0,0,0)
BABB	s_8	(2,0,0,0,0,0,0)
BABB	s_6	(6,0,0,0,0,0,0)
BABB	s_4	(8,0,0,0,0,0,0)
$s_{3.2}$		(1,0,0,1,0,0,0)
BABB	s_5	(4,0,0,0,0,0,0)
BABB	s_2	(2,0,0,4,0,0,0)
BABB	s_1	(10,0,0,8,0,0,0)
BBBB	s_5	(0,0,0,4,0,0,0)
$s_{3.3}$		(1,0,0,1,0,0,1)
BABB	s_7	(6,0,0,0,0,0,0)
BABB	s_9	(6,0,0,0,0,0,0)
BBBB	s_3	(4,0,0,4,0,0,2)
BBBB	s_7	(0,0,0,6,0,0,6)
BBBB	s_9	(0,0,0,6,0,0,6)
BBBB	s_{13}	(0,0,0,0,0,0,2)
$s_{3.4}$		(1,0,0,1,0,0,0)
BBBB	s_2	(4,0,0,2,0,0,0)
BABB	s_3	(4,0,0,4,0,0,0)
BBBB	s_1	(8,0,0,10,0,0,0)
$s_{3.5}$		(0,0,0,1,0,0,1)
BBBB	s_8	(0,0,0,2,0,0,0)
BBBB	s_6	(0,0,0,6,0,0,4)
BBBB	s_4	(0,0,0,8,0,0,10)
BBBB	s_{14}	(0,0,0,0,0,0,2)
$s_{3.6}$		(0,0,0,0,0,0,0)
BABB	s_{10}	(0,0,0,0,0,0,0)
BABB	s_{11}	(0,0,0,0,0,0,0)
BABB	s_{12}	(0,0,0,0,0,0,0)
BABB	s_{13}	(0,0,0,0,0,0,0)
BABB	s_{14}	(0,0,0,0,0,0,0)
BBBB	s_{10}	(0,0,0,0,0,0,0)
BBBB	s_{11}	(0,0,0,0,0,0,0)
BBBB	s_{12}	(0,0,0,0,0,0,0)

edge in the set of edges corresponding to state $s_{3.2}$ has weight

$$M^{(BABB)}\boldsymbol{w}(s_2) = [(1 \to 2,3,4),(4 \to 1)](4,0,0,2,0,0,0)^\top$$
$$= (2,0,0,4,0,0,0)^\top.$$

C. The Input Tagging

Now we establish the hypothesis of Theorem 4.1 for the graph $H^{(a)} = H^{(2)}$. For each of the 104 states t of $H^{(2)}$, we must find a matrix $R(t)$ as in the theorem. First we will argue the case of the state $s_{3.3}$ as a special case, and then make a general argument that covers all other states of $H^{(2)}$. What makes state $s_{3.3}$ special is that its weight has three nonzero entries; all other states of $H^{(2)}$ have weights with at most two nonzero entries.

Table XV lists the outgoing edges from state $s_{3.3}$ in $H^{(2)}$.

Using the edge label w listed in the first column and the terminal state $\tau(e)$ listed in the second column one can verify the edge weight $\boldsymbol{w}(e)$ listed in the third column by looking up the label matrix $M^{(w)}$ in Table X, and the state weight $\boldsymbol{w}(\tau(e))$ in the appropriate Table XI–XIV, and then calculating $\boldsymbol{w}(e) = M^{(w)}\boldsymbol{w}(\tau(e))$.

The last column lists a proposed input tagging $\mathcal{I}$ of these edges by the 16 input symbols 0, 1, $\cdots$, 9, a, b, c, d, e, f. This input tagging defines the matrix $R(s_{3.3})$ as follows. Order the outgoing edges from state $s_{3.3}$ as they are ordered in Table XV (so e_1 corresponds to the first row, and so on, up to e_{28}, which corresponds to the last row). Then the matrix $W(s_{3.3})$ is more-or-less the transpose of the third column of Table XV

$$W(s_{3.3}) = \begin{bmatrix} 1111110111111001111000000000000000000000 \\ 0000000000000000000000000000000000000000 \\ 0000000000000000000000000000000000000000 \\ 0000000000000000111101111111011111100000 \\ 0000000000000000000000000000000000000000 \\ 0000000000000000000000000000000000000000 \\ 0000000000000000101011111110011111110110 \end{bmatrix}$$
$$\mathcal{I}(\cdot) \quad 0123450678 9ab00cdef001234560789abcd0f70$$

If $R(s_{3.3})$ is defined by

$$R(s_{3.3})_{ij} = \begin{cases} 1, & \text{if row } i \text{ of Table XV has input symbol } j \\ 0, & \text{otherwise} \end{cases}$$

then

$$W(s_{3.3})R(s_{3.3}) = \begin{bmatrix} 1111111111111111 \\ 0000000000000000 \\ 0000000000000000 \\ 1111111111111111 \\ 0000000000000000 \\ 0000000000000000 \\ 1111111111111111 \end{bmatrix}$$
$$= \boldsymbol{w}(s_{3.3})[1\,1\,\cdots\,1],$$

which verifies the hypothesis of Theorem 4.1 for the state $s_{3.3}$.

It remains to verify the hypothesis of Theorem 4.1 for the other 103 states of $H^{(2)}$. We can recast the hypothesis in terms of the Set n-Coloring problem (see Section II). The set U corresponds to the set of edges $e_1, \cdots, e_{d_t}$ emanating from the state t of $H^{(a)}$, and hence corresponds to the set of columns of $W(t)$. The subsets $S_1, S_2, \cdots, S_m$ correspond to the real states $s_1, s_2, \cdots, s_m$ of $G^{(a)}$ that underlie state t, and hence they correspond to the nonzero rows of $W(t)$. Specifically we have $i \in S_k$ if and only if $W(t)_{ki} > 0$. To find a matrix $R(t)$ as in the theorem is to find a set coloring

$$\mathcal{I}\colon \{\text{columns of } W(t)\} \to \{0, 1, \cdots, n-1\}.$$

If there is only one set S_1 in the collection of sets to be colored (corresponding to a state of $H^{(a)}$ whose weight has only one nonzero entry), then a coloring exists if and only

TABLE XII
THE SPLITTING OF s_7 IN THE SECOND ROUND

$\mathsf{w}(s_7) = (0,0,6,0,0,6,0)$		
label	$\tau(e)$	$\mathsf{w}(e)$
$s_{7.1}$		(0,0,1,0,0,1,0)
CCAB	s_{13}	(0,0,2,0,0,0,0)
CCAB	s_{14}	(0,0,2,0,0,0,0)
CCCC	s_5	(0,0,4,0,0,0,0)
CCCA	s_5	(0,0,4,0,0,0,0)
CCAB	s_3	(0,0,2,0,0,4,0)
CCCA	s_2	(0,0,2,0,0,4,0)
CCCC	s_{11}	(0,0,0,0,0,2,0)
CCCC	s_{12}	(0,0,0,0,0,2,0)
CCCC	s_{10}	(0,0,0,0,0,4,0)
$s_{7.2}$		(0,0,1,0,0,1,0)
CCCA	s_8	(0,0,2,0,0,0,0)
CCCC	s_7	(0,0,6,0,0,6,0)
CCCC	s_4	(0,0,8,0,0,10,0)
$s_{7.3}$		(0,0,1,0,0,1,0)
CCCA	s_6	(0,0,6,0,0,0,0)
CCCA	s_7	(0,0,6,0,0,0,0)
CCCA	s_3	(0,0,4,0,0,4,0)
CCCC	s_{13}	(0,0,0,0,0,2,0)
CCAB	s_1	(0,0,0,0,0,10,0)
$s_{7.4}$		(0,0,1,0,0,1,0)
CCCA	s_9	(0,0,6,0,0,0,0)
CCAB	s_6	(0,0,4,0,0,6,0)
CCCC	s_6	(0,0,6,0,0,4,0)
CCCC	s_{14}	(0,0,0,0,0,2,0)
CCAB	s_5	(0,0,0,0,0,4,0)
$s_{7.5}$		(0,0,1,0,0,1,0)
CCAB	s_7	(0,0,6,0,0,6,0)
CCAB	s_4	(0,0,10,0,0,8,0)
CCAB	s_2	(0,0,0,0,0,2,0)
$s_{7.6}$		(0,0,1,0,0,1,0)
CCAB	s_9	(0,0,6,0,0,6,0)
CCCA	s_1	(0,0,10,0,0,8,0)
CCAB	s_8	(0,0,0,0,0,2,0)
$s_{7.7}$		(0,0,0,0,0,0,0)
CCCC	s_1	(0,0,0,0,0,0,0)
CCCC	s_2	(0,0,0,0,0,0,0)
CCCC	s_3	(0,0,0,0,0,0,0)
CCCC	s_8	(0,0,0,0,0,0,0)
CCCC	s_9	(0,0,0,0,0,0,0)
CCAB	s_{10}	(0,0,0,0,0,0,0)
CCAB	s_{11}	(0,0,0,0,0,0,0)
CCAB	s_{12}	(0,0,0,0,0,0,0)
CCCA	s_4	(0,0,0,0,0,0,0)
CCCA	s_{10}	(0,0,0,0,0,0,0)
CCCA	s_{11}	(0,0,0,0,0,0,0)
CCCA	s_{12}	(0,0,0,0,0,0,0)
CCCA	s_{13}	(0,0,0,0,0,0,0)
CCCA	s_{14}	(0,0,0,0,0,0,0)

TABLE XIII
THE SPLITTING OF s_9 IN THE SECOND ROUND

$w(s_9) = (0,0,0,6,0,0,6)$		
label	$\tau(e)$	$w(e)$
$s_{9.1}$		(0,0,0,1,0,0,0)
BBAB	s_{13}	(0,0,0,2,0,0,0)
BBCC	s_5	(0,0,0,4,0,0,0)
BBCA	s_5	(0,0,0,4,0,0,0)
BBCA	s_6	(0,0,0,6,0,0,0)
$s_{9.2}$		(0,0,0,1,0,0,1)
BBAB	s_{14}	(0,0,0,2,0,0,0)
BBCA	s_7	(0,0,0,6,0,0,0)
BBAB	s_3	(0,0,0,2,0,0,4)
BBCA	s_2	(0,0,0,2,0,0,4)
BBCA	s_3	(0,0,0,4,0,0,4)
BBCC	s_{10}	(0,0,0,0,0,0,4)
$s_{9.3}$		(0,0,0,1,0,0,1)
BBCA	s_8	(0,0,0,2,0,0,0)
BBCC	s_7	(0,0,0,6,0,0,6)
BBCC	s_4	(0,0,0,8,0,0,10)
$s_{9.4}$		(0,0,0,1,0,0,1)
BBCA	s_9	(0,0,0,6,0,0,0)
BBAB	s_6	(0,0,0,4,0,0,6)
BBCC	s_6	(0,0,0,6,0,0,4)
BBCC	s_{14}	(0,0,0,0,0,0,2)
BBAB	s_5	(0,0,0,0,0,0,4)
$s_{9.5}$		(0,0,0,1,0,0,1)
BBAB	s_7	(0,0,0,6,0,0,6)
BBAB	s_4	(0,0,0,10,0,0,8)
BBAB	s_2	(0,0,0,0,0,0,2)
$s_{9.6}$		(0,0,0,1,0,0,1)
BBAB	s_9	(0,0,0,6,0,0,6)
BBCA	s_1	(0,0,0,10,0,0,8)
BBAB	s_8	(0,0,0,0,0,0,2)
$s_{9.7}$		(0,0,0,0,0,0,1)
BBCC	s_{11}	(0,0,0,0,0,0,2)
BBCC	s_{12}	(0,0,0,0,0,0,2)
BBCC	s_{13}	(0,0,0,0,0,0,2)
BBAB	s_1	(0,0,0,0,0,0,10)
$s_{9.8}$		(0,0,0,0,0,0,0)
BBCC	s_1	(0,0,0,0,0,0,0)
BBCC	s_2	(0,0,0,0,0,0,0)
BBCC	s_3	(0,0,0,0,0,0,0)
BBCC	s_8	(0,0,0,0,0,0,0)
BBCC	s_9	(0,0,0,0,0,0,0)
BBAB	s_{10}	(0,0,0,0,0,0,0)
BBAB	s_{11}	(0,0,0,0,0,0,0)
BBAB	s_{12}	(0,0,0,0,0,0,0)
BBCA	s_4	(0,0,0,0,0,0,0)
BBCA	s_{10}	(0,0,0,0,0,0,0)
BBCA	s_{11}	(0,0,0,0,0,0,0)
BBCA	s_{12}	(0,0,0,0,0,0,0)
BBCA	s_{13}	(0,0,0,0,0,0,0)
BBCA	s_{14}	(0,0,0,0,0,0,0)

if S_1 has at least n elements. But this is true by the weight condition satisfied by $H^{(a)}$.

If there are two sets S_1 and S_2 to be colored (corresponding to a state of $H^{(a)}$ whose weight has two nonzero entries), then, again, there is a coloring if and only if S_1 and S_2 each have at least n elements. (First color their intersection, and use the remaining colors to color elements in their symmetric difference in any fashion.) Again, the fact that S_1 and S_2 are big enough follows from the weight condition on $H^{(a)}$.

It remains to show that all states of $H^{(2)}$ except state $s_{3.3}$ have weights with at most two nonzero entries. Referring to Table X, we see that all the states except state s_3 of $H^{(1)}$ have weights with at most two nonzero entries. Since the weight of a descendant state is at most the weight of its ancestor, all

TABLE XIV
THE SPLITTING OF s_{13} IN THE SECOND ROUND

$w(s_{13}) = (0,0,0,0,0,2,0)$		
label	$\tau(e)$	$w(e)$
$s_{13.1}$		(0,0,0,0,0,1,0)
ABBC	s_{11}	(0,0,0,0,0,2,0)
ABBC	s_{12}	(0,0,0,0,0,2,0)
ABBC	s_{13}	(0,0,0,0,0,2,0)
ABBC	s_{14}	(0,0,0,0,0,2,0)
ABBC	s_6	(0,0,0,0,0,4,0)
ABBC	s_{10}	(0,0,0,0,0,4,0)
$s_{13.2}$		(0,0,0,0,0,1,0)
ABBC	s_7	(0,0,0,0,0,6,0)
ABBC	s_4	(0,0,0,0,0,10,0)
$s_{13.3}$		(0,0,0,0,0,0,0)
ABBC	s_1	(0,0,0,0,0,0,0)
ABBC	s_2	(0,0,0,0,0,0,0)
ABBC	s_3	(0,0,0,0,0,0,0)
ABBC	s_5	(0,0,0,0,0,0,0)
ABBC	s_8	(0,0,0,0,0,0,0)
ABBC	s_9	(0,0,0,0,0,0,0)

TABLE XV
THE OUTGOING EDGES FROM STATE $s_{3.3}$ IN $H^{(2)}$

$w(s_{3.3}) = (1,0,0,1,0,0,1)$			
label	$\tau(e)$	$w(e)$	$\mathcal{I}(e)$
BABB	$s_{7.1}$	(1,0,0,0,0,0,0)	0
BABB	$s_{7.2}$	(1,0,0,0,0,0,0)	1
BABB	$s_{7.3}$	(1,0,0,0,0,0,0)	2
BABB	$s_{7.4}$	(1,0,0,0,0,0,0)	3
BABB	$s_{7.5}$	(1,0,0,0,0,0,0)	4
BABB	$s_{7.6}$	(1,0,0,0,0,0,0)	5
BABB	$s_{7.7}$	(0,0,0,0,0,0,0)	0
BABB	$s_{9.1}$	(1,0,0,0,0,0,0)	6
BABB	$s_{9.2}$	(1,0,0,0,0,0,0)	7
BABB	$s_{9.3}$	(1,0,0,0,0,0,0)	8
BABB	$s_{9.4}$	(1,0,0,0,0,0,0)	9
BABB	$s_{9.5}$	(1,0,0,0,0,0,0)	a
BABB	$s_{9.6}$	(1,0,0,0,0,0,0)	b
BABB	$s_{9.7}$	(0,0,0,0,0,0,0)	0
BABB	$s_{9.8}$	(0,0,0,0,0,0,0)	0
BBBB	$s_{3.1}$	(1,0,0,0,0,0,0)	c
BBBB	$s_{3.2}$	(1,0,0,1,0,0,0)	d
BBBB	$s_{3.3}$	(1,0,0,1,0,0,1)	e
BBBB	$s_{3.4}$	(1,0,0,1,0,0,0)	f
BBBB	$s_{3.5}$	(0,0,0,1,0,0,1)	0
BBBB	$s_{3.6}$	(0,0,0,0,0,0,0)	0
BBBB	$s_{7.1}$	(0,0,0,1,0,0,1)	1
BBBB	$s_{7.2}$	(0,0,0,1,0,0,1)	2
BBBB	$s_{7.3}$	(0,0,0,1,0,0,1)	3
BBBB	$s_{7.4}$	(0,0,0,1,0,0,1)	4
BBBB	$s_{7.5}$	(0,0,0,1,0,0,1)	5
BBBB	$s_{7.6}$	(0,0,0,1,0,0,1)	6
BBBB	$s_{7.7}$	(0,0,0,0,0,0,0)	0
BBBB	$s_{9.1}$	(0,0,0,1,0,0,0)	7
BBBB	$s_{9.2}$	(0,0,0,1,0,0,1)	8
BBBB	$s_{9.3}$	(0,0,0,1,0,0,1)	9
BBBB	$s_{9.4}$	(0,0,0,1,0,0,1)	a
BBBB	$s_{9.5}$	(0,0,0,1,0,0,1)	b
BBBB	$s_{9.6}$	(0,0,0,1,0,0,1)	c
BBBB	$s_{9.7}$	(0,0,0,0,0,0,1)	d
BBBB	$s_{9.8}$	(0,0,0,0,0,0,0)	0
BBBB	$s_{13.1}$	(0,0,0,0,0,0,1)	f
BBBB	$s_{13.2}$	(0,0,0,0,0,0,1)	7
BBBB	$s_{13.3}$	(0,0,0,0,0,0,0)	0

the descendants of these states other than state s_3 have at most two nonzero entries. Now referring to Table XI we see that the only descendant of state s_3 that has more than two nonzero entries is state $s_{3.3}$.

This completes the argument establishing the hypothesis of Theorem 4.1 for all states of $H^{(2)}$.

V. NP-Complete Problems in Constrained Code Design

In this section, motivated in part by the $(1,3;3)$-code constructions in the preceding two sections, we examine the complexity of certain general problems that arise in the construction of constrained codes. In particular, we will show that for a finite-state encoder graph, the general problem of finding a consistent input tag assignment that will ensure sliding-block decodability—the Encoder Input Tag Assignment—is NP-complete. We also show that the input tagging problem that arose in connection with the application of the matrix generalization of state-splitting is NP-complete. The connections of these problems to Graph n-Coloring and Set n-Coloring, respectively, are key to these proofs.

A thorough and illuminating introduction to the theory of NP-completeness may be found in [6]. For our purposes, the following streamlined and heuristic discussion will suffice.

The theory of NP-completeness has to do with the complexity of algorithms for solving decision problems. A decision problem is a general question applicable to each member of a class of objects. The objects are characterized by a set of parameters. For example, each object might be a finite set of points, $\{p_i | i = 1, \cdots, N\}$, together with the set of mutual distances $d(p_i, p_j)$, and a constant D. The question might be: Is there an ordering of the points, defined by a permutation π on the integers $\{1, 2, \cdots, N\}$, such that

$$\left(\sum_{i=1}^{N-1} d(p_{\pi(i)}, p_{\pi(i+1)})\right) + d(p_{\pi(N)}, p_{\pi(1)}) \leq D.$$

That is, the problem is to determine if there is a "tour" of the points, starting and ending at the point $p_{\pi(1)}$, and visiting each of the other points exactly once, such that the length of the tour does not exceed D.

An *instance* of the decision problem arises from the specification of particular values for the parameters used to characterize the objects. In the example, an instance would be a particular set of points, their mutual distances, and the constant D. One can make precise the notion of the size of an instance in terms of the length needed to specify the input parameters in some encoding scheme. For example, the mutual distances $d(p_i, p_j)$, $i < j$ and D above might be given in binary representation and the length would be the minimum number of bits required to do so.

The complexity of an algorithm for solving the decision problem can be measured in terms of the number of operations, or the execution time, required to execute the algorithm, using some model of computing, such as a Turing machine model. An algorithm is considered to be efficient if the complexity is a polynomial function of the size of an instance. A problem is said to be in the class P if there exists such a solution algorithm of polynomial-time complexity. A problem is deemed to be intractable if there is provably no efficient algorithm for determining a solution. Among the known intractable problems are those which are undecidable—for which there is no algorithm, efficient or otherwise, to determine a solution—as well as those which are decidable by some algorithm. The latter also have the property that, roughly speaking, there is provably no efficient algorithm for simply verifying whether a proposed solution answers the decision problem.

There are many decision problems, however, for which no efficient solution algorithm is known, but for which there exists an efficient verification algorithm. A problem of this kind is said to be in class NP, indicating that it can be solved in polynomial time by a nondeterministic algorithm (an algorithm that, roughly speaking, evaluates all possible solutions simultaneously). It is evident that the class P is a subset of the class NP. The fundamental unsolved problem of complexity theory is whether the class P is a *proper* subset of the class NP.

Within the class NP, certain problems, known as NP-complete problems, have been shown to have the following property: the existence of a polynomial-time algorithm for solving any NP-complete problem would imply that P = NP. On the other hand, if any NP-complete problem is intractable, then every NP-complete problem is intractable. In some sense, then, the NP-complete problems may be viewed as the most difficult problems in the class NP.

Given a decision problem that belongs to NP, it is useful to know if it is, in fact, NP-complete. One can determine this by demonstrating a polynomial-time reduction of any instance of a particular NP-complete problem to an instance of the problem in question. In the remainder of this section, we demonstrate that certain problems related to input tag assignments and sliding-block decodability are NP-complete.

A. NP Completeness of Encoder Input Tag Assignment

Let $\mathcal{E}$ be an (S, n)-encoder graph. In this section, we examine the complexity of determining the existence of an input tag assignment such that the resulting encoder is (m, a)-sliding-block-decodable. In particular, we will prove the following result.

Theorem 5.1: Let $\mathcal{E}$ be an (S, n) encoder. Given nonnegative integers m and a, the problem of deciding whether there exists an n-ary input tag assignment such that the tagged encoder is (m, a)-sliding-block-decodable is solvable in polynomial time for $n = 2$, but is NP-complete for $n \geq 3$.

The key step in the proof of the theorem is the transformation of the graph n-coloring problem to an instance of the (m, a)-sliding-block decodability problem. We then invoke the following known facts about the n-coloring problem.

For the case $n = 2$, there is a polynomial-time solution to the problem which actually produces a 2-coloring if one exists. The following algorithm should be applied to each irreducible component of the given graph H. We denote the two colors as $\{0, 1\}$. Each step of the algorithm will make use of a designated state that we will call a *hub state*.

1) Choose an initial hub state v arbitrarily and assign a color $P(v)$.
2) For all states u such that u, v determines an edge in H, assign the complementary color, i.e., $P(u) = \overline{P(v)}$.
3) If the assigned color at any state conflicts with a previous assignment for the state, STOP. The graph is not 2-colorable.
4) Else choose a new hub state v from among the already colored states, excluding those which were a previously designated hub state. If no such state exists, STOP. The assignment P is a 2-coloring.
5) Else, go to step 2).

For $n \geq 3$, the graph n-colorability problem is known to be NP-complete [6].

First, we reduce the general sliding-block decodability problem to the problem of block decodability.

Lemma 5.2: The (m, a)-sliding-block decodability problem is polynomially equivalent to the $(0,0)$-sliding-block-decodability problem, i.e., the block-decodability problem.

Proof: Let G be a labeled graph with edge label function L for which we wish to find an (m, a)-sliding-block-decodable input tag assignment. We first assign to each (m, a)-connectedness equivalence class a unique symbol, $l[e]$. Note that the (m, a)-adjacency of pairs of edges can be determined by looking at the edges in the graph $\boldsymbol{H}(G; m, a)$, defined in Section III-A, whose construction requires only polynomial time. Now (m,a)-connectedness can be computed in polynomial time as the equivalence relation induced by the (m,a)-adjacency relation. We now define a new edge label function L' by setting $L'(e) = l[e]$ for all edges e. This labeling satisfies the property that two edges e, f in G share the same label $L'(e) = L'(f)$ if and only if $[e] = [f]$; that is, they are (m, a)-connected with respect to the original labeling L. It follows immediately that an input tag assignment for G is (m, a)-sliding-block-decodable with respect to L if and only if it is block-decodable with respect to L'. □

The block-decodability problem is easily seen to be in NP, so Theorem 5.1 will follow from the following proposition.

Proposition 5.3: Let G be an undirected graph. There is an (S, n) encoder graph $\mathcal{E}$ with the following properties:

1) $\mathcal{E}$ is obtained from G by a polynomial-time modification.
2) $\mathcal{E}$ has a block-decodable input tag assignment if and only if G is n-colorable.

Proof: Let $e_0, \cdots, e_{N-1}$ denote the $N = |E|$ edges in G. The encoder graph $\mathcal{E}$ has a distinct state v_i corresponding to each edge e_i. If, in G, $e = e_i$ has initial state $s = \sigma(e)$ and terminal state $t = \tau(e)$, the state v_i in $\mathcal{E}$ has two outgoing edges with terminal state $v_{(i+1) \bmod (N)}$ and labels s and t, respectively. Each state v_i is then assigned an additional $n-2$ edges with terminal states chosen arbitrarily. The labels on these added edges are chosen to be distinct from one another as well as all previously assigned edge labels. The resulting graph $\mathcal{E}$ is an irreducible, (S, n) encoder for the constrained system $S(\mathcal{E})$. From the construction, it is clear that any block-decodable input tag assignment provides an n-coloring of G. Conversely, any n-coloring of G gives a partial assignment of input tags to the edges derived from the states of G. At each state of $\mathcal{E}$, an arbitrary assignment of the unused input tags to the untagged outgoing edges yields a block-decodable input tag assignment. □

Remark: Since Subgraph Encoder Input Tag Assignment has Encoder Input Tag Assignment as a special case, the theorem implies that this more general problem is NP-hard.

B. NP Completeness of Input Tagging in the Matrix Construction

In Section IV-A, we introduced a matrix generalization of state-splitting that was then applied to the construction of a rate 4 : 8, $(0, 2)$-sliding-block-decodable $(1, 3; 3)$ encoder. In general, the construction of a $(0, a)$-sliding-block-decodable encoder using this approach requires the specification of an n-ary input tag assignment for a graph $G^{(a)}$ underlying a polygraph $H^{(a)}$ obtained after a rounds of generalized state-splittings. In this section, we prove that this input tagging problem is, in general, NP-complete. We formally state the problem as follows.

Polygraph n-ary Input Tag Assignment: Given a polygraph $H^{(a)}$ and its associated weight matrices $W(t)$, one for each state t of $H^{(a)}$, as specified in Theorem 4.1, do there exist matrices $R(t)$ satisfying the hypotheses of Theorem 4.1?

Theorem 5.4: Polygraph n-ary Input Tag Assignment is NP-complete for $n \geq 2$.

The proof proceeds in two steps. We first prove that Set n-Coloring, the problem corresponding to the verification of the hypothesis of Theorem 4.1, is NP-complete. We also show that even for fixed n, Set n-Coloring is NP-complete for $n \geq 2$. We then use this fact to prove that the consistent input tag assignment problem for the graph produced by the matrix-based construction is NP-complete.

Recall the statement of Set n-Coloring, reproduced here for convenience.

Set n-Coloring: Given an integer $n \geq 1$, and given a collection of subsets $S_1, S_2, \cdots, S_m$ of a finite set U, is there a coloring

$$\mathcal{I}\colon \bigcup_{k=1}^{m} S_k \rightarrow \{0, 1, \cdots, n-1\}$$

such that: for each $1 \leq k \leq m$, and for each color $0 \leq j \leq n-1$, the set S_k has at least one element of color j?

We remark that, if we impose the additional constraint that each set S_i, $1 \leq k \leq m$ has cardinality equal to two, then Set 2-Coloring is solvable by a polynomial algorithm. Specifically, we can define a graph G with vertex set given by the set U, and edge set given by the collection of subsets S_i, $1 \leq k \leq m$. The set 2-coloring problem is simply the 2-coloring problem for the graph G, and we have described a polynomial solution to this problem in Section V-A.

Theorem 5.5: Set n-Coloring is NP-complete. For fixed n, Set n-Coloring is NP-complete for $n \geq 2$.

Proof: It is not difficult to show that a proposed coloring can be checked in polynomial time, so Set n-Coloring is

NP. We will show that Set n-Coloring is NP-complete by reducing the NP-complete problem Satisfiability (or SAT) [6] to it. Recall that a *truth assignment* to a set $\{x_1, \cdots, x_m\}$ of Boolean variables is a function $\tau : \{x_1, \cdots, x_m\} \to \{0, 1\}$. A *clause* is a disjunction (or) of Boolean variables and negations of Boolean variables, such as $x_1 \vee \overline{x}_3 \vee x_8$. (One can regard the logical operators as $\overline{x} = 1-x$ and $x_1 \vee x_2 = \max(x_1, x_2)$ if one likes.) A clause is *satisfied* by a truth assignment τ if and only if it evaluates to 1 (true) when each variable x_i is replaced by its assignment, $\tau(x_i) \in \{0, 1\}$. Thus the clause $x_1 \vee \overline{x}_3 \vee x_8$ is satisfied by τ if and only if $\tau(x_1) = 1$, or $\tau(x_3) = 0$, or $\tau(x_8) = 1$.

Here is Satisfiability:

Satisfiability (SAT): Given a set V of Boolean variables, and a collection C of clauses over V, is there a truth assignment τ that satisfies all of the clauses in C?

(We remark that SAT was the original NP-complete problem [6].)

Now we reduce Satisfiability to Set 2-Coloring, showing that Set 2-Coloring is NP-complete. We remark that there is an easy reduction of Set n-Coloring to Set $(n+1)$-Coloring, so it will follow that Set n-Coloring is NP-complete for $n \geq 2$.

We are given an instance (V, C) of Satisfiablity. Implement this instance as an instance of Set 2-Coloring as follows. The universal set U contains the Boolean variables $x_1, \cdots, x_m$, their complements $\overline{x}_1, \cdots, \overline{x}_m$, and two additional elements denoted t and f. We define a subset $S_0 = \{t, f\}$ and, for each Boolean variable, we define a subset $S_i = \{x_i, \overline{x}_i\}$. Finally, for each clause $C_i = v_i^1 \vee v_i^2 \vee \cdots \vee v_i^{k_i}$ in C, where each v_i^j represents a Boolean variable or its complement, we define a subset $T_i = \{f, v_i^1, v_i^2, \cdots, v_i^{k_i}\}$.

Now, suppose we are given a valid set 2-coloring. We define a truth assignment according to the rule: $\tau(x_i) = 1$ if and only if $\mathcal{I}(x_i) = \mathcal{I}(t)$. It is easy to see that the properties of the set 2-coloring ensure that this assignment τ satisfies all of the clauses in C. Conversely, given a truth assignment that satisfies all of the clauses in C, we can use the same rule to specify a set 2-coloring for the collection of subsets $\{S_i\}$, $1 \leq k \leq m$. It follows that Set 2-Coloring is NP-complete. □

We can now show that Polygraph n-ary Input Tag Assignment is NP-complete, for $n \geq 2$. Verifying that a proposed matrix $R(t)$ is a solution is a polynomial algorithm, so the problem is in the class NP. We show the problem is NP-complete by reducing Set n-Coloring to it.

Let $(U, S_1, S_2, \cdots, S_m)$ be an instance of Set n-Coloring. We construct a constraint graph $G^{(0)}$ as follows. For each set S_i, the graph $G^{(0)}$ has a state that we call S_i. For each $x_j \in S_i$, the state S_i of has a distinct outgoing edge e with label $L(e) = x_j$. To ensure that $G^{(0)}$ is irreducible, we resort to the following artifice. For each set S_i, linearly order its outgoing edges and declare that the first edge terminates at state S_{i+1}, and declare that the others terminate at S_i.

Construct the polygraph $H^{(0)}$ as in Section IV-A. Set the initial weight vector $\boldsymbol{w}(s_0) = (1, 1, \cdots, 1)$.

The matrix $W = W(s_0)$ becomes the matrix whose rows are indexed by $S_1, \cdots, S_m$ and whose columns are indexed by the elements of $\cup_i S_i$. The entry $W(S_i, x_j)$ is 1 or 0 according to whether or not $x_j \in S_i$. A matrix R (whose rows are indexed by the x_j and whose columns are indexed by the input symbols $0, 1, \cdots, n-1$) having elementary rows that solves

$$WR \geq \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} [1 \quad 1 \quad \cdots \quad 1]$$

gives a solution to the Set n-Coloring problem by setting $\mathcal{I}(x_i) = j$ if the x_ith row of R has its single 1 in column j. This completes the reduction and the proof of Theorem 5.4. □

C. Deciding when there is a 1-Block Decoder from an Edge System is NP-Complete

The *edge system* presented by a directed graph G is the system presented by G when the label $L(e)$ of an edge e of G is just e itself: $L(e) = e$. Thus the symbols of the edge system are just the edges of G, and the allowed symbol sequences are just the paths of G. We denote the edge system of G by $\boldsymbol{X}_G$.

If there is an $(\boldsymbol{X}_G, n)$ encoder having a 1-block decoder, then we can form a subgraph G' of G and an input tagging of the edges of G' as follows: if the 1-block decoder decodes the label e (an edge of G) to the symbol j, then (and only then) we declare e to be an edge of G' and we assign the tag $\mathcal{I}(e) = j$ to the edge e. Since any tag sequence can be encoded into a label sequence in $\boldsymbol{X}_G$ using the $(\boldsymbol{X}_G, n)$ encoder, there is for each tag sequence $i_0 i_1 \cdots i_k$ a path $e_0 e_1 \cdots e_k$ in G' that generates it as $\mathcal{I}(e_0 e_1 \cdots e_k) = i_0 i_1 \cdots i_k$. In other words, the 1-block map defined by $\mathcal{I}$ is onto the n-shift.

We make the further assumptions that G' is an irreducible graph, and that the capacity of the edge system $\boldsymbol{X}_{G'}$ is $\log(n)$. In this case, we can apply the following theorem (a special case of a result of Coven and Paul) [5] to conclude that the tagging of G' by $\mathcal{I}$ is *lossless*: any two distinct paths in G' that share the same initial state and share the same terminal state generate distinct tag sequences using the tagging function $\mathcal{I}$.

Theorem 5.6 [5]: Let $\phi: \boldsymbol{X}_G \to S$ be an onto 1-block map $\phi: \boldsymbol{X}_G \to S$ from an irreducible edge system $\boldsymbol{X}_G$ to a sofic system S, defined by a labeling L of the edges of G by the symbols of S. Then $\mathbf{cap}(\boldsymbol{X}_G) = \mathbf{cap}(S)$ if and only if the labeling L is lossless.

This discussion suggests the following problem which, for future reference, we call the *1-block surjection problem for the n-shift.*

1-Block Surjection for the n-Shift: Given an integer $n \geq 2$ and a directed graph G with $\mathbf{cap}(\boldsymbol{X}_G) \geq n$, is there an irreducible subgraph $G' \subseteq G$ with $\mathbf{cap}(\boldsymbol{X}_{G'}) = n$ and a 1-block map from $\boldsymbol{X}_{G'}$ onto the n-shift?

In this section, we address the complexity of 1-Block Surjection. Specifically, we prove the following NP-completeness result.

Theorem 5.7: 1-Block Surjection for the n-Shift is NP-complete. For fixed n, it is NP-complete for $n \geq 2$.

Proof: We use Theorem 5.6 to show that the problem is in NP. To verify that a proposed tagging $\mathcal{I}$ of G' by the input symbols $0, 1, \cdots, n-1$ defines a mapping *onto* the full

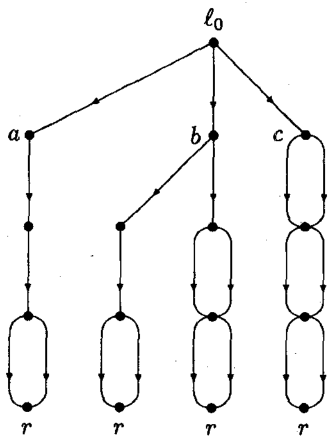

Fig. 12. The subgraph of paths from the leaf l_0 to the root r in G.

n-shift, we must verify that the tagging is lossless. This can be done in polynomial time by constructing the fiber product $G' * G'$ and checking whether there are any paths in $G' * G'$ that leave the diagonal and then return to it.

To show that the problem is NP-complete, we reduce Set n-Coloring to 1-Block Surjection for the n-Shift. Our proof is for the case $n = 2$. The proof for general n is not really any more difficult. We handle the specific case of $n = 2$ for clarity.

We are given an instance $(U, S_1, S_2, \cdots, S_m)$ of set 2-coloring. U is a finite set, and each S_j is a subset of U. We are to answer the following question: Is there a coloring function

$$\mathcal{I}\colon \bigcup_{j=1}^{m} S_j \to \{0, 1\}$$

such that for each $1 \le k \le m$, and for each color $i \in \{0, 1\}$, the set S_k has at least one element of color i?

Construct a directed graph G as follows. G has a special state r which we call the *root* state. Construct a directed binary tree rooted at state r as follows. The edges of the tree are directed away from the root. Each internal (nonleaf) state of the tree has two outgoing edges to states in the tree at the next depth. Construct the tree in any (definite) way so that it has exactly $m + 1$ leaves. For example, one could always grow only the leftmost branch, creating a very unbalanced tree. Or one could grow all the branches, creating a complete tree of sufficient depth. It does not matter, as long as the tree is grown following a definite procedure that is polynomial in the length of the instance of Set 2-Coloring.

Enumerate the tree's leaves as $l_0, l_1, \cdots, l_m$. From leaf l_0, construct the subgraph as shown in Fig. 12. Notice that there are $2^4 = 16$ paths of length 4 from l_0 to r. We will shortly construct further incoming edges to states b and c of Fig. 12. State a of Fig. 12 will have no more incoming edges.

Enumerate the elements of $\cup_{i=1}^{m} S_i$ as $x_1, x_2, \cdots, x_k$. For each x_j, G has a state x_j and a single edge from x_j to r. For each set S_i, G has a state S_i. For each element $x_j \in S_i$, G has a single edge from state S_i to state x_j.

For each leaf l_i, $1 \le i \le m$, construct the following subgraph (see Fig. 13). Form a path of length 2 starting at the leaf l_i, passing through a new state a_i, and ending at the

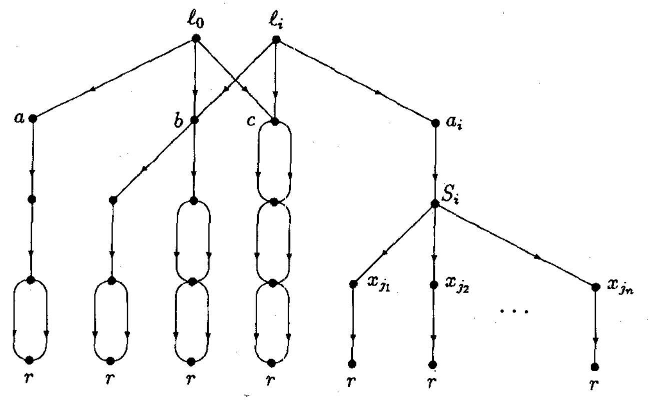

Fig. 13. The subgraph of paths from leaves l_0 and l_i to the root r in G.

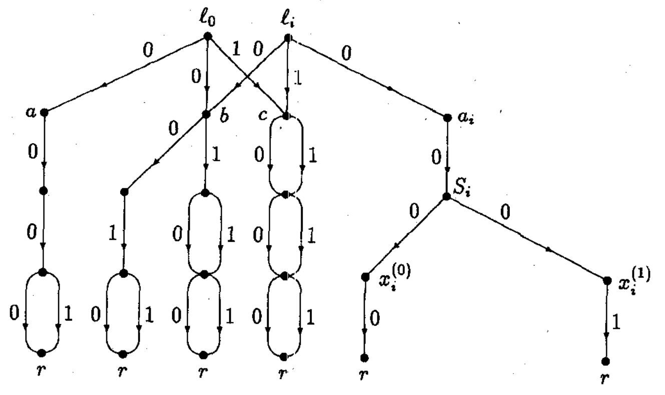

Fig. 14. The tagged paths from leaves l_0 and l_i to the root r in G'.

state S_i. Add an edge from l_i to state b and an edge from l_i to state c. This completes the construction of the graph G.

First suppose that the instance $(U, S_1, \cdots, S_m)$ of set 2-coloring has a solution $\mathcal{I}\colon \cup_i S_i \to \{0, 1\}$. We form the subgraph G' of G as follows. For each $1 \le i \le m$, choose $x_i^{(0)}, x_i^{(1)} \in S_i$ with $\mathcal{I}(x_i^{(0)}) = 0$ and $\mathcal{I}(x_i^{(1)}) = 1$. Delete from G all of the states x_j (and their incident edges) except those that are chosen as $x_i^{(0)}$ or $x_i^{(1)}$ for some $1 \le i \le m$. The remaining graph is G'. Note that G' is irreducible because every path in G' must pass through r in a time bounded by 4 plus the depth of the binary tree.

We tag the edges along the paths of length 4 emanating from the leaf states l_0 and l_i in G' as shown in Fig. 14. In particular, note that the tags on the edges leaving the states $x_i^{(0)}$ and $x_i^{(1)}$ are well-defined as $\mathcal{I}(x_i^{(0)}) = 0$ and $\mathcal{I}(x_i^{(1)}) = 1$, respectively. Also notice that the $2^4 = 16$ paths of length 4 emanating from each leaf state (and ending at r) are distinctly tagged by the 16 binary strings of length 4.

Finally, for each internal state of the binary tree rooted at state r, distinctly tag its two outgoing edges (with 0 and 1).

We show that this tagging defines a 1-block map from $X_{G'}$ onto the 2-shift. Call a nonempty path in G' that starts at state r, does not pass through r, and ends at r, a *return*. The reader can verify that for any sufficiently long binary string $\boldsymbol{w}$, there is a return in G' that is tagged by a nonempty prefix $\boldsymbol{w}'$ of $\boldsymbol{w}$. It follows from this and an induction that the 1-block map defined by the tagging is onto the 2-shift.

The tagging of G' is lossless because for each state s of G', any two paths of length 3 emanating from s that generate the same tag sequence actually share the same initial edge. It follows from Theorem 5.6 that

$$\mathbf{cap}\,(\boldsymbol{X}_{G'}) = \mathbf{cap}\,(\boldsymbol{X}_2) = \log(2).$$

Now suppose that there is an irreducible subgraph G' of G with $\mathbf{cap}\,(\boldsymbol{X}_{G'}) = \log(2)$ and a 1-block map from $\boldsymbol{X}_{G'}$ onto the 2-shift. This 1-block map is lossless by Theorem 5.6.

For each state S_i, $1 \leq i \leq m$, of the original graph G, all the paths of length 2 emanating from S_i end at state r. It follows that for each such state S_i that remains in the subgraph G', S_i has out-degree at most $2^2 = 4$; otherwise, G' would not admit a lossless tagging. Similarly, for each state l_i of G, all the paths of length 4 emanating from l_i end at state r. It follows that for each such state l_i that remains in the subgraph G', there are at most $2^4 = 16$ paths of length 4 in G' emanating from state l_i. We denote the number of paths of length 4 emanating from state l_i as $n(l_i)$. Note $n(l_i) \leq 16$.

Because each internal state of the binary tree in G rooted at r has out-degree 2, we have

$$\sum_{\text{leaves } l \text{ in } G} 2^{-\text{depth}(l)} = 1.$$

So

$$\sum_{\text{leaves } l \text{ in } G'} n(l) 2^{-\text{depth}\,(l)-4} = \sum_{\text{leaves } l \text{ in } G'} \frac{n(l)}{16} 2^{-\text{depth}\,(l)} \leq 1$$

and the inequality is strict if either G' has fewer leaves than G has or for some leaf l of G', $n(l) < 16$. But $\mathbf{cap}\,(\boldsymbol{X}_{G'}) = \log(2)$, so

$$\sum_{\text{leaves } l \text{ in } G'} n(l) 2^{-\text{depth}\,(l)-4} = \sum_{\text{returns } p \text{ in } G'} 2^{-\text{length}\,(p)} = 1.$$

It follows that all leaves in G remain in G' and there are exactly 16 paths of length 4 in G' from each leaf to state r. Referring to Fig. 12 and using that $n(l_0) = 16$, we see that there are exactly six paths of length 3 in G' from state b to state r, and exactly eight paths of length 3 in G' from state c to state r. In the subgraph G', for each $1 \leq i \leq m$, there are at most four paths of length 3 from state a_i to state r. But $n(l_i) = 16$, so all three of the edges emanating from state l_i in G must remain in the subgraph G'. It follows that there are exactly $16 - 8 - 6 = 2$ paths of length 3 in G' from state a_i to state r. Thus for each $1 \leq i \leq m$, the state S_i remains in G'; moreover, the state has exactly two outgoing edges in G'. Denote as $x_{j(i)}$ and $x_{j'(i)}$ the terminal states of these two edges. (Recall, in terms of the set 2-coloring problem, $x_{j(i)}, x_{j'(i)} \in S_i$.)

We can assume, without loss of generality, that the paths of length 4 in G' from state l_i through states b and c are tagged as they are in Fig. 14. The tags of these paths account for all of the binary sequences of length 4 except the two sequences 0000 and 0001. Thus these two remaining sequences are generated by the two paths of length 4 in G' that emanate from state l_i and pass through states $x_{j(i)}$ and $x_{j'(i)}$. As a consequence, the edges leaving these two states are distinctly tagged. Finally, define a set 2-coloring $\mathcal{I}\colon \cup_i S_i \to \{0, 1\}$ by first defining $\mathcal{I}(x_{j(i)})$ and $\mathcal{I}(x_{j'(i)})$ to be these distinct tags, and then extending $\mathcal{I}$ arbitrarily to all of $\cup_i S_i$. This completes the proof. □

VI. Conclusions

In this paper, we presented two detailed and quite different constructions of rate $1/2$, sliding-block codes for the runlength-limited, charge-constrained system with parameters $(d, k; c) = (1, 3; 3)$. These represent the first 100% efficient, sliding-block codes for this constraint to appear in the literature. Heuristics for designing sliding-block-decodable encoders using state-splitting and a new matrix generalization of state-splitting were also developed and illustrated by means of the constructions.

We established relationships between the general problem of determining the existence of sliding-block-decodable input tag assignments for finite-state encoder graphs (and subgraphs) and combinatorial problems of graph colorability and set colorability. Exploiting these connections, we considered complexity issues associated with these problems of coding and combinatorics and proved NP-completeness results for several of them.

Acknowledgment

The authors wish to thank B. Marcus for very helpful discussions during the course of this research. They would also like to thank the anonymous reviewer for clarifying the exposition and simplifying the proofs of Theorems 2.1, 5.1, and 5.5.

References

[1] 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.
[2] J. J. Ashley and B. H. Marcus, "Canonical encoders for sliding block decoders," *SIAM J. Discr. Math.*, vol. 8, pp. 555–605, 1995.
[3] J. Ashley, M. Hilden, P. Perry, and P. Siegel, "Correction to 'A note on the Shannon capacity of run-length-limited codes'," *IEEE Trans. Inform. Theory*, vol. 39, no. 3, pp. 1110–1112, May 1993.
[4] J. Ashley and P. Siegel, "A note on the Shannon capacity of run-length-limited codes," *IEEE Trans. Inform. Theory*, vol. IT-33, no. 4, pp. 601–605, July 1987.
[5] E. Coven and M. Paul, "Endomorphisms of irreducible shifts of finite type," *Math. Syst. Theory*, vol. 8, pp. 167–175, 1974.
[6] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: W. H. Freeman, 1979.
[7] R. Karabed and B. H. Marcus, "Sliding-block coding for input-restricted channels," *IEEE Trans. Inform. Theory*, vol. 34, no. 1, pp. 2–26, Jan. 1988.
[8] R. Karabed and P. H. Siegel, "A 100% efficient sliding-block code for the charge-constrained, runlength-limited channel with parameters $(d, k; c) = (1, 3; 3)$," in *Proc. 1991 IEEE Int. Symp. on Information Theory* (Budapest, Hungary, 1991), p. 229.
[9] D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding.* New York: Cambridge Univ. Press, 1995.
[10] B. H. Marcus, "Factors and extensions of full shifts," *Monats. Math*, vol. 88, pp. 239–247, 1979.
[11] ——, "Sofic systems and encoding data," *IEEE Trans. Inform. Theory*, vol. IT-31, no. 3, pp. 366–377, May 1985.
[12] B. Marcus, R. Roth, and P. Siegel, "Constrained systems and coding for recording channels," in *Handbook of Coding Theory.* Amsterdam, The Netherlands: Elsevier, 1996, to be published.

[13] B. H. Marcus, P. H. Siegel, and J. K. Wolf, "Finite-state modulation codes for data storage," *IEEE J. Sel. Areas Commun.*, vol. 10, no. 1, pp. 5–37, Jan. 1992.

[14] A. M. Patel, "Zero-modulation encoding in magnetic recording," *IBM J. Res. Devel.*, vol. 19, no. 4, pp. 366–378, July 1975.

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

[16] P. H. Siegel, "On the complexity of limiting error propagation in sliding block codes," in *Proc. 2nd IBM Symp. on Coding and Error Control* (San Jose, CA, Jan. 1985).