Chapter 5

The State-Splitting Algorithm

In this chapter, we provide an exposition of the state-splitting algorithm, which implements the proof of Theorem 4.1, for constructing finite-state encoders. The steps in the algorithm are summarized in Figure 5.9.

The approach we will follow uses graph construction techniques, based on state splitting and approximate eigenvectors, which have their roots in symbolic dynamics, where they were introduced by R.F. Williams [Will73] and Adler, Goodwyn, and Weiss [AGW77]. The first application of state-splitting ideas in constrained coding was Patel's construction of the Zero-Modulation (ZM) code [Patel75] (see Section 4.5). The state-splitting algorithm is also related to earlier ideas of Franaszek [Fra80b], [Fra82], [Fra89].

For a given deterministic presentation G of a constrained system S and an achievable rate $p/q \leq \mathsf{cap}(S)$, we will apply a state-splitting transformation iteratively beginning with the qth power graph G^q; the procedure culminates in a new presentation of S^q with minimum out-degree at least 2^p; then, after deleting edges, we get an $(S^q, 2^p)$-encoder, which, when tagged, gives our desired rate $p : q$ finite-state encoder for S.

Although the design procedure can be made completely systematic—in the sense of having the computer automatically generate an encoder and decoder for any valid code rate—the application of the method to just about any nontrivial code design problem will benefit from the interactive involvement of the code designers. There are some practical tools that can help the designer make "good" choices during the construction process. We will discuss some of these tools in Section 5.5.

It should be stressed that the general problem of designing codes that achieve, for example, the minimum number of encoder states, minimum sliding-block decoding window, or the less precise feature of minimum hardware complexity, is not solved. This remains an active research topic, as exemplified by recent papers where lower bounds on the number of encoder states [MR91] and the minimum sliding-block decoder window are studied [Ash88], [Kam89], [Imm92], [Holl95], [AM97], [AM00]. See Chapters 6 and 7 for more on this.

5.1 State splitting

In this section, we define state splitting of a labeled graph H and later apply it to $H = G^q$. We begin with a simplified special case.

Let $H = (V, E, L)$ be a labeled graph and denote by E_u the set of outgoing edges from state u in H. A *basic out-splitting* at state u is determined by a partition

$$E_u = E_u^{(1)} \cup E_u^{(2)}$$

of E_u into two disjoint sets. This partition is used to define a new labeled graph $H' = (V', E', L')$ that changes the local picture at state u. The set of states V' consists of all states $v \neq u$ in H, as well as two new states denoted $u^{(1)}$ and $u^{(2)}$:

$$V' = (V - \{u\}) \cup \{u^{(1)}, u^{(2)}\} .$$

The states $u^{(1)}$ and $u^{(2)}$ are called *descendant states* of state u, and state u is called the *parent state* of $u^{(1)}$ and $u^{(2)}$.

The edges in H' that do not involve states $u^{(1)}$ and $u^{(2)}$ are inherited from H. That is, if there is an edge e from state v to state v' in H, (with $v, v' \neq u$) there is a corresponding edge in H'. For edges involving state u, we consider the following three cases.

Case 1: Let edge e in H start at a state $v \neq u$ and terminate in state u. This edge is replicated in H' to produce two edges: an edge $e^{(1)}$ from v to $u^{(1)}$ and an edge $e^{(2)}$ from v to $u^{(2)}$.

Case 2: Let edge e in H start at state u and terminate in a state $v \neq u$, and suppose e belongs to the set $E_u^{(i)}$ in the partition of E_u. We draw in H' a corresponding edge from state $u^{(i)}$ to state v.

Case 3: Let edge e be a self-loop at state u in H, and suppose that e belongs to $E_u^{(i)}$. In H' there will be two edges from state $u^{(i)}$ corresponding to e: one edge to state $u^{(1)}$, the other to state $u^{(2)}$.

As with states, we refer to *descendant edges* in H' and *parent edges* in H. In all cases, the edge label of an edge in H' is the edge label of its parent edge in H.

In specifying the partitions in particular examples, we will refer to the edges by their edge labels in cases where this causes no ambiguity.

The change in the local picture at state u is shown in Figures 5.1 and 5.2. In the figures, we have partitioned the set of edges E_u into subsets, $E_u^{(1)} = \{a, b\}$ and $E_u^{(2)} = \{c\}$. The state u splits into two states, $u^{(1)}$ and $u^{(2)}$, according to the partition. It is evident that the anticipation at states v_1 and v_2 may increase by one symbol. So, H' need not be deterministic even if H is.

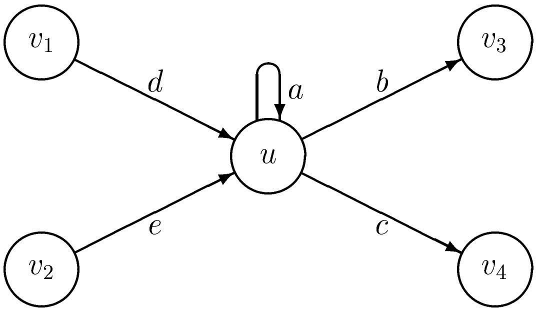

Figure 5.1: Local picture at state u before splitting.

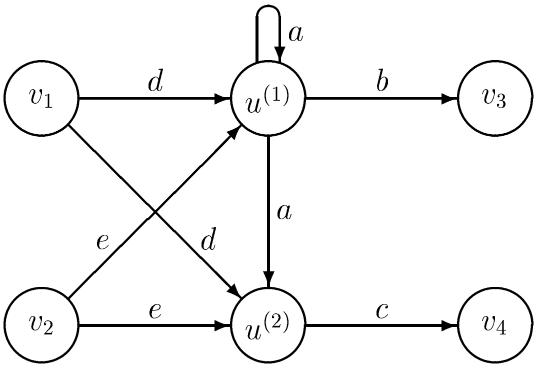

Figure 5.2: Basic out-splitting at state u for Figure 5.1.

In general, a state splitting may involve partitions into any number of subsets, and several states may be split simultaneously; so, we have the following more general notion of state splitting.

An *out-splitting* of a labeled graph H begins with a partition of the set, E_u, of outgoing edges for each state u in H into $N(u)$ disjoint subsets

$$E_u = E_u^{(1)} \cup E_u^{(2)} \cup \cdots \cup E_u^{(N(u))} .$$

From the partition, we derive a new labeled graph H'. The set of states $V_{H'}$ consists of the descendant states $u^{(1)}, u^{(2)}, \ldots, u^{(N(u))}$ for every $u \in V_H$. Outgoing edges from state u in H are partitioned among its descendant states and replicated in H' to each of the descendant terminal states as follows: for each edge e from u to v in H, determine the partition element $E_u^{(i)}$ to which e belongs, and endow H' with edges $e^{(r)}$ from $u^{(i)}$ to $v^{(r)}$ for $r = 1, 2, \ldots, N(v)$; the label of $e^{(r)}$ in H' is the same as the label of e in H.

Sometimes an out-splitting is called a *round* of out-splitting to indicate that several states may have been split simultaneously.

The labeled graph H' obtained from H by out-splitting is sometimes called an out-splitting of H. It has several important characteristics, relative to H, enumerated in the following proposition.

Proposition 5.1 *Let H be a labeled graph and let H' be obtained from H by out-splitting. Then*

1. *$S(H') = S(H)$.*
2. *If H has anticipation $\mathcal{A}$, then H' has anticipation at most $\mathcal{A}+1$.*
3. *If H is (m, a)-definite, then H' is $(\mathsf{m}, \mathsf{a}+1)$-definite.*
4. *If H is irreducible, so is H'.*

The key to this result is the following fact, which is a consequence of the definition of out-splitting.

Lemma 5.2 *Let H be a labeled graph and let H' be obtained from H by out-splitting. Then $e_1^{(r_1)} e_2^{(r_2)} \ldots e_\ell^{(r_\ell)}$ is a path in H' if and only if $e_1 e_2 \ldots e_\ell$ is a path in H and*

$$e_{i+1} \in E_{\tau(e_i)}^{(r_i)} \quad \textit{for} \quad i = 1, 2, \ldots, \ell{-}1 \, .$$

Moreover, both paths generate the same word.

We leave the proof of the lemma to the reader.

Proof of Proposition 5.1. 1. This follows immediately from Lemma 5.2.

2. Let $e_1^{(r_1)} e_2^{(r_2)} \ldots e_{\mathcal{A}+2}^{(r_{\mathcal{A}+2})}$ and $\hat{e}_1^{(s_1)} \hat{e}_2^{(s_2)} \ldots \hat{e}_{\mathcal{A}+2}^{(s_{\mathcal{A}+2})}$ be paths of length $\mathcal{A}+2$ in H' starting at the same state and generating the same word. Then $e_1 e_2 \ldots e_{\mathcal{A}+2}$ and $\hat{e}_1 \hat{e}_2 \ldots \hat{e}_{\mathcal{A}+2}$ are paths in H that start at the same initial state and generate the same word. Thus $e_1 = \hat{e}_1$ and $e_2 = \hat{e}_2$. Since e_2 belongs to the partition element $E_{\tau(e_1)}^{(r_1)}$ and $\hat{e}_2$ belongs to the partition element $E_{\tau(\hat{e}_1)}^{(s_1)} = E_{\tau(e_1)}^{(s_1)}$, it then follows that $r_1 = s_1$. So, $e_1^{(r_1)} = \hat{e}_1^{(s_1)}$. Thus, H' has anticipation at most $\mathcal{A}+1$.

3. The proof of this is similar to 2 and is left to the reader.

4. Let $u^{(s)}$ and $v^{(t)}$ be states in H' and let $\gamma = e_1 e_2 \ldots e_\ell$ be any path in H from u to v such that $e_1 \in E_u^{(s)}$. Then, by Lemma 5.2, there is a path $\gamma' = e_1^{(r_1)} e_2^{(r_2)} \ldots e_\ell^{(r_\ell)}$ in H' where

the r_i are determined by $e_{i+1} \in E_{\tau(e_i)}^{(r_i)}$ for $i = 1, 2, \ldots \ell{-}1$ and $r_\ell = t$. Observe that γ' is a path from $u^{(s)}$ to $v^{(t)}$ in H'. $\square$

The *complete out-splitting* of a labeled graph H is the out-splitting obtained from the partition in which each set in the partition consists of a single, distinct edge. The resulting graph is exactly the Moore co-form of H, as was defined in Section 2.2.7.

We also have the notion of *in-splitting* obtained by reversing the roles of outgoing and incoming edges in the definition of out-splitting.

Finally, we mention that the out-splitting graph transformation can be described in terms of adjacency matrices as follows. A 0–1 matrix is called a *division matrix* if it has exactly one 1 in each column and at least one 1 in each row. Now, given an out-splitting H' of H, let D be the division matrix with rows indexed by states of H and columns indexed by states of H', defined by

$$D_{u,v^{(i)}} = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{if } u \neq v \end{cases} ,$$

and let C be the matrix with rows indexed by states of H' and columns indexed by states of H, defined by

$$C_{v^{(i)},u} = \text{number of edges in } E_v^{(i)} \text{ which terminate in } u \, .$$

Then one can check that

$$A_H = DC \quad \text{and} \quad A_{H'} = CD \, .$$

Conversely, if there is a division matrix D and a matrix C with nonnegative integer entries such that $A_H = DC$ and $A_{H'} = CD$, then H' is an out-splitting of H; we leave the proof of this to the reader.

5.2 Approximate eigenvectors and consistent splitting

The state-splitting algorithm that we will present starts with a deterministic graph presentation of a given constrained system S and, through a sequence of rounds of out-splitting, ends up with an (S, n)-encoder. One key question to answer is which states to split and how to split them. Approximate eigenvectors, to be discussed next, serve as such a guide. In fact, as we show in Section 5.6 and Chapter 7, approximate eigenvectors are useful not only for the synthesis of finite-state encoders, but also for analyzing them.

5.2.1 Approximate eigenvectors

Given a nonnegative integer square matrix A and an integer n, an (A, n)-*approximate eigenvector* is a nonnegative integer vector $\mathbf{x} \not\equiv \mathbf{0}$ such that

$$A\mathbf{x} \geq n\mathbf{x} \ ,$$

where the (weak) inequality holds componentwise. We refer to this inequality as the *approximate eigenvector inequality*. The set of all (A, n)-approximate eigenvectors is denoted $\mathcal{X}(A, n)$.

When A is the adjacency matrix of a graph G, the approximate eigenvector inequality has a very simple meaning in terms of G. Think of the vector $\mathbf{x} = (x_u)_{u \in V_G}$ as assigning *state weights*: the weight of state u is x_u. Now assign *edge weights* to the edges of the graph according to their terminal states: the weight of an edge e is given by $x_{\tau_G(e)}$. Recalling that E_u denotes the set of outgoing edges from state u in G, the approximate eigenvector inequality can be written as the set of simultaneous scalar inequalities, one for each state u,

$$\sum_{e \in E_u} x_{\tau_G(e)} \geq n x_u \quad \text{for every } u \in V_G \ .$$

That is, the sum of the weights of the outgoing edges from a given state u is at least n times the weight of the state u itself.

Example 5.1 Let G be the presentation of the $(0,1)$-RLL constrained system shown in Figure 2.2. The third power of G is shown in Figure 2.10, and the adjacency matrix $A_{G^3} = A_G^3$ of G^3 satisfies

$$A_G^3 \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix} \geq 4 \begin{pmatrix} 2 \\ 1 \end{pmatrix} .$$

Therefore, the vector $\mathbf{x} = (2\ 1)^\top$ is an $(A_{G^3}, 4)$-approximate eigenvector. □

The following result is straightforward.

Proposition 5.3 *For a graph G, the all-one vector $\mathbf{1}$ is an (A_G, n)-approximate eigenvector if and only if G has minimum out-degree at least n. Also, a 0–1 vector is an (A_G, n)-approximate eigenvector if and only if G has a subgraph with minimum out-degree at least n.*

The next result tells us that approximate eigenvectors exist when we need them.

Theorem 5.4 *Let A be a nonnegative integer square matrix and let n be a positive integer. Then*

$$\mathcal{X}(A,n) \neq \emptyset \quad \textit{if and only if} \quad \lambda(A) \geq n \ .$$

Furthermore, if A is irreducible and $\lambda(A) = n$, then every (A,n)-approximate eigenvector is a right eigenvector associated with the eigenvalue n.

Proof. *Sufficiency:* Assume that $\lambda(A) \geq n$. We first show that there is an (A,n)-approximate eigenvector under the assumption that A is irreducible. We distinguish between the following two cases.

Case 1: $\lambda(A) > n$. By Theorem 3.11(b), A has a strictly positive right eigenvector $\mathbf{y}$ associated with $\lambda(A)$. We first perturb the entries of $\mathbf{y}$ to obtain a new vector $\hat{\mathbf{y}}$, with positive rational entries, that satisfies the inequality $A\hat{\mathbf{y}} \geq n\hat{\mathbf{y}}$. Now, let $\mathbf{x}$ be the vector obtained from $\hat{\mathbf{y}}$ by clearing denominators—i.e., by multiplying $\hat{\mathbf{y}}$ by a common multiple of the denominators of its entries. The vector $\mathbf{x}$ has positive integer entries, and it satisfies the approximate eigenvector inequality since $\hat{\mathbf{y}}$ does. Thus, $\mathbf{x}$ is an (A,n)-approximate eigenvector.

Case 2: $\lambda(A) = n$. Since $\lambda(A) = n$ is an eigenvalue of A, there is a nontrivial solution (i.e., *not* the zero vector) to the homogeneous linear system of equations

$$(A - nI)\mathbf{y} = \mathbf{0} \ .$$

Since the coefficients of this linear system are rational numbers, we can assume, by applying Gaussian elimination, that $\mathbf{y}$ has rational entries. Clearing denominators, we obtain a solution $\mathbf{x}$ with integer entries. By Theorem 3.11(d) it follows that this solution is an integer eigenvector associated with $\lambda(A) = n$, and by Theorem 3.11(b) it is strictly positive (possibly after multiplying each of its entries by -1). Hence, $\mathbf{x}$ is a positive integer eigenvector and, as such, it is an (A,n)-approximate eigenvector. This completes the proof of sufficiency in case A is irreducible.

If A is a $k \times k$ reducible matrix, then there is, by Theorem 3.15(a), an irreducible component B of A with $\lambda(B) = \lambda(A) \geq n$. Thus, by what we have just proved, there is a (B,n)-approximate eigenvector $\mathbf{x}$. We extend $\mathbf{x}$ to a vector with k entries simply by setting to zero the entries that are indexed by columns of A that do not contain columns of B. This new vector is an (A,n)-approximate eigenvector.

Necessity: Suppose first that A is irreducible and let $\mathbf{x}$ be an (A,n)-approximate eigenvector. Also, let $\mathbf{z}$ be a left eigenvector associated with the eigenvalue $\lambda = \lambda(A)$. By Theorem 3.11(b), the eigenvector $\mathbf{z}$ can be assumed to be strictly positive. Hence,

$$\lambda\mathbf{z}\mathbf{x} = \mathbf{z}\lambda\mathbf{x} = \mathbf{z}A\mathbf{x} \geq n\mathbf{z}\mathbf{x} \ .$$

Noting that $\mathbf{z}\mathbf{x} > 0$, we thus obtain $\lambda \geq n$, with equality ($\lambda = n$) if and only if $A\mathbf{x} = n\mathbf{x}$. Therefore, if $\lambda = n$, every (A,n)-approximate eigenvector is a right eigenvector associated with the eigenvalue n.

Finally, suppose that A is reducible and let $\tilde{A}$ be the matrix obtained by removing the rows and columns of A that contain the irreducible components of A whose columns all index zero components in $\mathbf{x}$. Let B be an irreducible sink in $\tilde{A}$ and let $\mathbf{y}$ be the subvector of $\mathbf{x}$ which is indexed by the columns of B in A. It can be readily verified that the vector $\mathbf{y}$ is a (B, n)-approximate eigenvector and, so, $n \le \lambda(B) \le \lambda(\tilde{A}) \le \lambda(A)$. □

5.2.2 Computing approximate eigenvectors

In this section, we describe an algorithm for computing (A, n)-approximate eigenvectors. The algorithm is due to Franaszek [Fra82, Appendix] (see also [ACH83, Appendix]), and its running time is proportional to the *values* (rather than the size of the bit representations) of the computed approximate eigenvector [MR91].

The Franaszek algorithm is presented in Figure 5.3. The input to the algorithm is a

---

$\mathbf{y} \leftarrow \boldsymbol{\xi}$;
$\mathbf{x} \leftarrow \mathbf{0}$;
while ($\mathbf{x} \ne \mathbf{y}$) {
  $\mathbf{x} \leftarrow \mathbf{y}$;
  $\mathbf{y} \leftarrow \min\left\{\left\lfloor \frac{1}{n} A\mathbf{x} \right\rfloor, \mathbf{x}\right\}$;   /* apply $\lfloor\cdot\rfloor$ and $\min\{\cdot,\cdot\}$ componentwise */
}
return $\mathbf{x}$;

Figure 5.3: Franaszek algorithm for computing (A, n)-approximate eigenvectors.

nonnegative integer square matrix A, a positive integer n, and a nonnegative integer vector $\boldsymbol{\xi}$. The output is a nonnegative integer vector $\mathbf{x}$, the properties of which are summarized in Proposition 5.5(b) below.

For a nonnegative integer square matrix A, a positive integer n, and a nonnegative integer vector $\boldsymbol{\xi} = (\xi_u)_u$, let $\mathcal{X}(A, n; \boldsymbol{\xi})$ denote the set of all elements $\mathbf{x} = (x_u)_u$ of $\mathcal{X}(A, n)$ that are dominated by $\boldsymbol{\xi}$ (i.e., $x_u \le \xi_u$ for all u, or, in short, $\mathbf{x} \le \boldsymbol{\xi}$). Also, for a vector $\mathbf{y} = (y_u)_u$, we will use the notations $\|\mathbf{y}\|_1$ and $\|\mathbf{y}\|_\infty$ for $\sum_u |y_u|$ and $\max_u |y_u|$, respectively.

Proposition 5.5 *Let A be a nonnegative integer square matrix and let n be a positive integer.*

(a) If $\mathbf{x}, \mathbf{x}' \in \mathcal{X}(A, n)$, then the vector defined by $Bigl(\max(x_u, x'_u)\Big)_u$ belongs to $\mathcal{X}(A, n)$. Thus, for any nonnegative integer vector $\boldsymbol{\xi}$ there is a largest (componentwise) element of $\mathcal{X}(A, n; \boldsymbol{\xi})$ (provided of course that $\mathcal{X}(A, n; \boldsymbol{\xi}) \ne \emptyset$).

(b) The Franaszek algorithm eventually halts for any input vector $\boldsymbol{\xi}$; and the output is either the zero vector (if $\mathcal{X}(A, n; \boldsymbol{\xi}) = \emptyset$) or the largest (componentwise) element of $\mathcal{X}(A, n; \boldsymbol{\xi})$.

Proof. Part (a) is a straightforward computation. As for part (b), let $\mathbf{x}$ be an element of $\mathcal{X}(A, n; \boldsymbol{\xi})$ and let $\mathbf{y}_m$ denote the value of $\mathbf{y}$ at the beginning of the mth iteration of the main loop of the algorithm. We show inductively on m that $\mathbf{x} \le \mathbf{y}_m$. Clearly, this holds for $m = 1$, where we have $\mathbf{y}_1 = \boldsymbol{\xi}$. Now, assuming that $\mathbf{x} \le \mathbf{y}_m$, we also have $\mathbf{x} \le \frac{1}{n} A\mathbf{x} \le \frac{1}{n} A\mathbf{y}_m$. Since $\mathbf{x}$ is an integer vector we obtain

$$\mathbf{x} \le \min \left\{ \left\lfloor \frac{1}{n} A\mathbf{y}_m \right\rfloor, \mathbf{y}_m \right\} = \mathbf{y}_{m+1} .$$

It remains to show that the algorithm halts and produces the required vector $\mathbf{x}$. Assume first that $\mathbf{y}_{m+1} \neq \mathbf{y}_m$. Recalling that $\mathbf{y}_{m+1}$ is dominated by $\mathbf{y}_m$, we must have $\|\mathbf{y}_{m+1}\|_1 < \|\mathbf{y}_m\|_1$. Now, all vectors involved are nonnegative integer vectors and, therefore, the algorithm must eventually halt with $\mathbf{y}_{m+1} = \mathbf{y}_m$. At this point we have $\mathbf{y}_m = \mathbf{y}_{m+1} \le \left\lfloor \frac{1}{n} A\mathbf{y}_m \right\rfloor$. Hence, either $\mathbf{y}_m \in \mathcal{X}(A, n; \boldsymbol{\xi})$ or $\mathbf{y}_m = \mathbf{0}$. Furthermore, if $\mathcal{X}(A, n; \boldsymbol{\xi}) \neq \emptyset$, we must have $\mathbf{y}_m \neq \mathbf{0}$, as $\mathbf{y}_m$ dominates any vector in $\mathcal{X}(A, n; \boldsymbol{\xi})$. $\square$

By the proof of Proposition 5.5, it also follows that the number of iterations of the main loop of the algorithm is at most $\|\boldsymbol{\xi}\|_1 + 1$.

Now, suppose that A is a nonnegative integer $k \times k$ matrix and we would like to compute the vector in $\mathcal{X}(A, n)$ with the smallest norm $\|\mathbf{x}\|_\infty$. In order to find such a vector, we apply the Franaszek algorithm to vectors of the form $\boldsymbol{\xi} = \xi \cdot \mathbf{1}$ (namely, to integer multiples of the all-one vector), searching for the smallest positive integer ξ for which the algorithm produces a nonzero vector $\mathbf{x}$. By Theorem 5.4, there exists such ξ if $n \le \lambda(A_G)$. Performing a binary search on ξ, the number of integer operations thus totals to $O\left(k^2 \cdot \|\mathbf{x}\|_\infty \log \|\mathbf{x}\|_\infty\right)$. We point out, however, that there are families of $k \times k$ matrices A for which the computed vectors $\mathbf{x}$ are such that $\|\mathbf{x}\|_\infty$ is *exponential* in k. Such a family is described in Example 7.1.

Example 5.2 Let G denote the third power of the Shannon cover of the $(1, 7)$-RLL constrained system. The adjacency matrix of G is given by

$$A_G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} .$$

We apply the Franaszek algorithm to A_G and $n = 4$, with vectors of the form $\boldsymbol{\xi} = \xi \cdot \mathbf{1}$. The smallest value of ξ that results in a nonzero output is $\xi = 3$, in which case the output of the algorithm is

$$\mathbf{x} = (2\ 3\ 3\ 3\ 2\ 2\ 2\ 1)^\top .$$

We can now try to find other $(A_G, 4)$-approximate eigenvectors whose largest entry is 3 by applying the Franaszek algorithm with $\boldsymbol{\xi} = \mathbf{x}_i$ for $i = 0, 1, 2, \ldots, 7$, where

$$(\mathbf{x}_i)_u = \begin{cases} (\mathbf{x})_u & \text{if } u \neq i \\ (\mathbf{x})_u - 1 & \text{if } u = i \end{cases} .$$

That is, in each application of the algorithm, we start with a vector $\boldsymbol{\xi}$ obtained by subtracting 1 from one of the entries of $\mathbf{x}$. When doing so, we find that $\boldsymbol{\xi} = \mathbf{x}_7$ is the only case for which the algorithm produces a nonzero output: that output is $\mathbf{x}_7$ itself, namely,

$$(2\ 3\ 3\ 3\ 2\ 2\ 2\ 0)^\top .$$

It follows that there are only two $(A_G, 4)$-approximate eigenvectors, $\mathbf{x}$ and $\mathbf{x}_7$, whose largest entry is 3. □

Example 5.3 Let G be now the second power of the Shannon cover of the $(2, 7)$-RLL constrained system. By applying the Franaszek algorithm we find that $\mathcal{X}(A_G, 2; \xi \cdot \mathbf{1})$ is nonempty if and only if $\xi \geq 4$. The output of the algorithm for $\xi = 4$ is

$$\mathbf{x} = (2\ 3\ 4\ 4\ 3\ 3\ 2\ 1)^\top .$$

There is another $(A_G, 2)$-approximate eigenvector,

$$(2\ 3\ 4\ 4\ 3\ 3\ 1\ 1)^\top ,$$

whose largest entry is 4. □

Recall from Section 4.4 that a necessary and sufficient condition for the existence of a rate $p : q$ deterministic encoder for a constrained system S is the existence of a set of principal states; the definition of such a set (given in Section 4.4) depends on p, q and a deterministic presentation G of S (recall also that this condition was necessary for the existence of a block decodable encoder, in particular the existence of a block encoder). From Proposition 5.3, it is evident that such a set exists if and only if there is an $(A_G^q, 2^p)$-approximate eigenvector with 0–1 entries. Thus, the question of existence of a rate $p : q$ deterministic encoder can be answered by applying the Franaszek Algorithm to the vector $\boldsymbol{\xi} = \mathbf{1}$: such an encoder exists if and only if the algorithm does not return the zero vector.

5.2.3 x-consistent splitting

Let H be a labeled graph and let $\mathbf{x} = (x_v)_{v \in V_H}$ be an (A_H, n)-approximate eigenvector. A *basic* $\mathbf{x}$*-consistent partition* at state u is a partition of E_u into

$$E_u = E_u^{(1)} \cup E_u^{(2)} ,$$

with the property that

$$\sum_{e \in E_u^{(1)}} x_{\tau(e)} \geq n\, y^{(1)} \qquad \text{and} \qquad \sum_{e \in E_u^{(2)}} x_{\tau(e)} \geq n\, y^{(2)} ,$$

where $y^{(1)}$ and $y^{(2)}$ are positive integers and

$$y^{(1)} + y^{(2)} = x_u .$$

The out-splitting determined by this partition is called a *basic* $\mathbf{x}$-*consistent splitting* at state u, and we denote the resulting labeled graph by H'. It is straightforward to check that the induced vector $\mathbf{x}' = (x'_v)_v$, indexed by the states of H', defined by

$$x'_v = \begin{cases} x_v & \text{if} \quad v \neq u \\ y^{(1)} & \text{if} \quad v = u^{(1)} \\ y^{(2)} & \text{if} \quad v = u^{(2)} \end{cases} ,$$

is an $(A_{H'}, n)$-approximate eigenvector.

The cube of the $(0,1)$-RLL graph presentation is shown in Figure 5.4 (which is identical to Figure 2.10). Figure 5.5 shows the result of a basic $\mathbf{x}$-consistent splitting for Figure 5.4, with respect to the $(A_G^3, 2^2)$-approximate eigenvector $\mathbf{x} = (2\ 1)^\top$. State 0 is split into two descendant states, $0^{(1)}$ and $0^{(2)}$, according to the partition $E_0^{(1)} = \{011, 110, 010\}$ and $E_0^{(2)} = \{101, 111\}$. The induced vector is $\mathbf{x}' = (1\ 1\ 1)^\top$, and the resulting labeled graph therefore has minimum out-degree at least $2^2 = 4$.

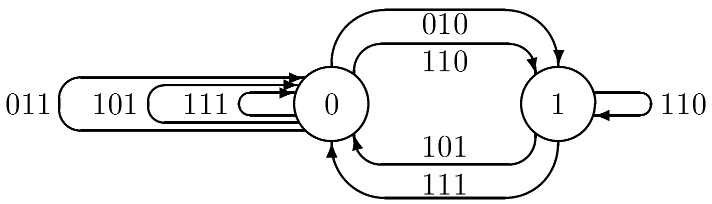

Figure 5.4: Cube of $(0,1)$-RLL graph presentation.

The notion of $\mathbf{x}$-consistency can be extended to out-splittings in general as follows.

Given a labeled graph H, a positive integer n, and an (A_H, n)-approximate eigenvector $\mathbf{x} = (x_v)_{v \in V_H}$, an $\mathbf{x}$-*consistent partition* of H is defined by partitioning the set, E_u, of outgoing edges for each state u in H into $N(u)$ disjoint subsets

$$E_u = E_u^{(1)} \cup E_u^{(2)} \cup \cdots \cup E_u^{(N(u))} ,$$

such that

$$\sum_{e \in E_u^{(r)}} x_{\tau(e)} \geq n x_u^{(r)} \qquad \text{for} \qquad r = 1, 2, \ldots, N(u) , \tag{5.1}$$

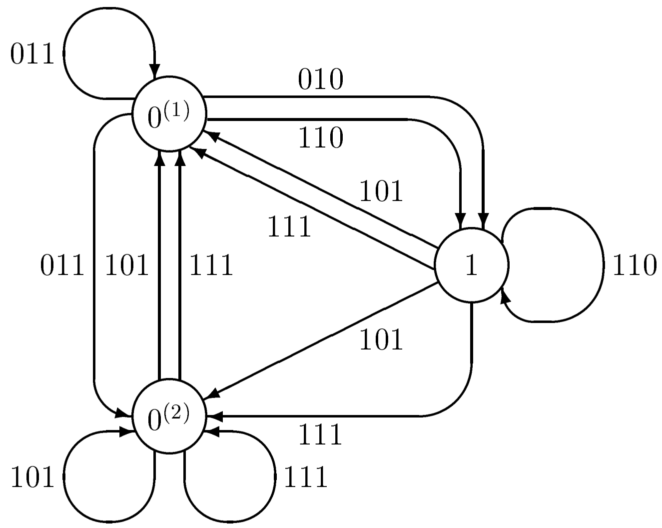

Figure 5.5: Basic $\mathbf{x}$-consistent splitting for Figure 5.4.

where $x_u^{(r)}$ are nonnegative integers and

$$\sum_{r=1}^{N(u)} x_u^{(r)} = x_u \quad \text{for every } u \in V_H. \tag{5.2}$$

The out-splitting based upon such a partition is called an $\mathbf{x}$-*consistent splitting*. The vector $\mathbf{x}'$ indexed by the states $u^{(r)}$ of the split graph H' and defined by $x'_{u^{(r)}} = x_u^{(r)}$ is called the *induced vector*.

An $\mathbf{x}$-consistent partition or splitting is called *non-trivial* if for at least one state u, $N(u) \geq 2$ and $x_u^{(1)}$ and $x_u^{(2)}$ are positive. Observe that any basic $\mathbf{x}$-consistent splitting is a non-trivial $\mathbf{x}$-consistent splitting.

Figures 5.6 and 5.7 give an example of an $\mathbf{x}$-consistent splitting in which two states, 0 and 1, are split simultaneously. An $(A_G, 2)$-approximate eigenvector is $\mathbf{x} = (2\ 2\ 1)^\top$ (in this particular case, it is actually an eigenvector). An $\mathbf{x}$-consistent splitting for states 0 and 1 can be carried out as follows. State 0 splits according to the partition $E_0^{(1)} = \{a\}$ and $E_0^{(2)} = \{b, c\}$. State 1 splits, simultaneously, according to the partition $E_1^{(1)} = \{d\}$ and $E_1^{(2)} = \{e\}$. This yields a new labeled graph H' with induced vector $\mathbf{x}' = (1\ 1\ 1\ 1\ 1)^\top$, and the resulting labeled graph therefore has minimum out-degree at least 2.

We summarize in Proposition 5.6 the important features of $\mathbf{x}$-consistent out-splittings.

Proposition 5.6 *Let H be a labeled graph and let $\mathbf{x}$ be an (A_H, n)-approximate eigenvector. Suppose that H' is obtained from H by an $\mathbf{x}$-consistent splitting and $\mathbf{x}'$ is the induced vector. Then*

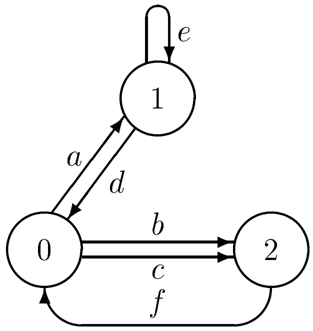

Figure 5.6: Labeled graph to be split.

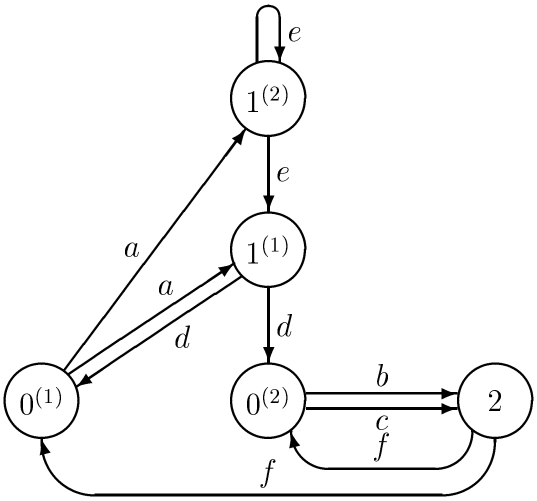

Figure 5.7: $\mathbf{x}$-consistent splitting for Figure 5.6.

1. *$\mathbf{x}'$ is an $(A_{H'}, n)$-approximate eigenvector.*

2. $\sum_{u \in V_H} x_u = \sum_{v \in V_{H'}} x'_v$.

Proof. 1. The inequality (5.1) says precisely that $(A_{H'}\mathbf{x}')_{u^{(r)}} \geq n\, x'_{u^{(r)}}$ for each state $u^{(r)}$ of H'. So, $\mathbf{x}'$ is an $(A_{H'}, n)$-approximate eigenvector.

2. This follows immediately from (5.2). □

5.3 Constructing the encoder

The key result needed for the construction of our encoders is as follows.

Proposition 5.7 *Let H be an irreducible labeled graph and assume that the all-one vector* $\mathbf{1}$ *is not an (A_H, n)-approximate eigenvector. Let $\mathbf{x}$ be a strictly positive (A_H, n)-approximate eigenvector. Then, there is a basic $\mathbf{x}$-consistent splitting of H.*

Before giving the proof, we describe how we will make use of it in an iterative fashion to construct finite-state encoders with finite anticipation.

Let G be a deterministic labeled graph presenting S and let p and q be integers such that $p/q \leq \mathsf{cap}(S)$. So, G^q is a deterministic labeled graph presenting S^q. Let $\mathbf{x} = (x_v)_{v \in V_G}$ be an $(A_G^q, 2^p)$-approximate eigenvector (which exists by Theorem 5.4). If $\mathbf{x}$ is a 0–1 vector, then some subgraph of G^q has minimum out-degree at least 2^p, and we are done. So, we may assume that there is no 0–1 $(A_G^q, 2^p)$-approximate eigenvector. Let G' be the labeled subgraph of G^q corresponding to the states of G with nonzero entries of $\mathbf{x}$. The vector $\mathbf{x}'$ obtained by restricting $\mathbf{x}$ to the states in G' is a strictly positive $(A_{G'}, 2^p)$-approximate eigenvector.

If G' is irreducible, then we will be in a position to apply Proposition 5.7 for $H = G'$ and $n = 2^p$. Otherwise, we can restrict to a sink G_0 of G'; recall that a sink is an irreducible component all of whose outgoing edges terminate in the component, and recall that every graph has a sink. Since G_0 is an irreducible component of G', and, by assumption, there is no 0–1 $(A_G^q, 2^p)$-approximate eigenvector, it follows that $\mathbf{1}$ is not an $(A_{G_0}, 2^p)$-approximate eigenvector. Moreover, since G_0 is a sink, it follows that the restriction, $\mathbf{x}_0$, of $\mathbf{x}'$ to G_0 is a strictly positive $(A_{G_0}, 2^p)$-approximate eigenvector. Proposition 5.7 can now be applied to carry out a basic $\mathbf{x}_0$-consistent splitting of G_0, producing an irreducible labeled graph G_1.

By Proposition 5.6, a basic $\mathbf{x}_0$-consistent splitting decomposes an entry of $\mathbf{x}_0$ into strictly smaller positive integers. So, iteration of this state-splitting procedure will produce a sequence of labeled graphs $G_1, G_2, \ldots, G_t$, where the graph G_t has an adjacency matrix A_{G_t} with an all-1's (A_{G_t}, n)-approximate eigenvector. Therefore, by Proposition 5.3, the graph G_t has minimum out-degree at least $n = 2^p$. Since, by Proposition 5.1, out-splitting preserves finite anticipation, the graph G_t has finite anticipation. Deleting excess edges, we pass to a subgraph of G_t' which is an $(S^q, 2^p)$-encoder. Now, tag this encoder with input labels, and we have our rate $p : q$ finite-state encoder for S with finite anticipation.

Having now completely described the construction of the encoder, we have completed the proof of Theorem 4.1 modulo the proof of Proposition 5.7.

Note that the number of iterations required to arrive at the encoder graph is no more than $\sum_{v \in V_G} (x_v - 1)$, since a state v with entry x_v will be split into at most x_v descendant states throughout the whole iteration process. So, by Proposition 5.1, the anticipation of G_t is at most $\sum_{v \in V_G} (x_v - 1)$. For the same reason, the number of states in the encoder graph is at most $\sum_{v \in V_G} x_v$.

If we delete the self-loop at state 1 and assign input tags to the labeled graph in Figure 5.5, we obtain a rate $2 : 3$ finite-state $(0, 1)$-RLL encoder shown in Figure 5.8. We indicate, for this example, how the encoding and decoding is implemented.

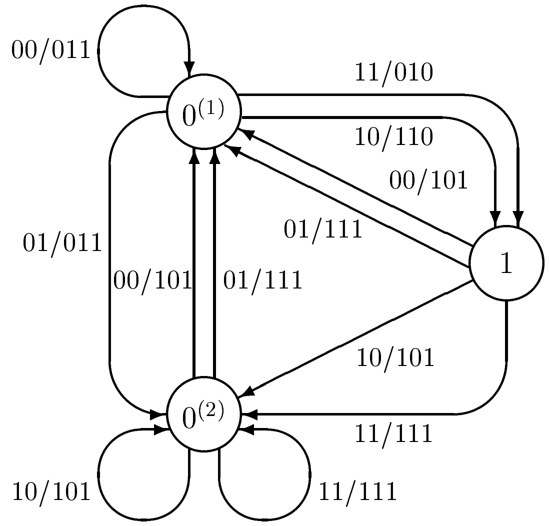

Figure 5.8: Tagged $(0, 1)$-RLL encoder.

If we initialize to state $0^{(1)}$, the data sequence of 2-blocks, 00 10 10 11, encodes to the $(0, 1)$-RLL sequence of 3-codewords

$$011\ 110\ 101\ 111\ .$$

We decode the $(0, 1)$-RLL codeword sequence just generated. Starting at state $0^{(1)}$, the edge determined by the codeword 011, with upcoming codeword 110, is the self-loop $0^{(1)} \stackrel{00/011}{\longrightarrow} 0^{(1)}$, so the decoder will generate the input tag 00. Proceeding, the codeword 110, with upcoming word 101, determines the edge $0^{(1)} \stackrel{10/110}{\longrightarrow} 1$. The reader can decode the next codeword 101 in a similar matter, and that is as far as we can go without knowing more upcoming codewords.

As we will see, the encoder of Figure 5.8 is not the smallest (in terms of number of states) rate $2 : 3$ finite-state $(0, 1)$-RLL encoder. We now proceed with the proof of Proposition 5.7.

Proof of Proposition 5.7. Let $x_{\max} \neq 1$ be the maximum of the entries of $\mathbf{x}$. We will show that there is a state u with the following properties:

$$x_u = x_{\max} \tag{5.3}$$

and

$$(A_H)_{u,v} \neq 0 \quad \text{for some state } v \text{ with } x_v < x_{\max}\ . \tag{5.4}$$

We then show that there is a basic $\mathbf{x}$-consistent splitting at each such state u.

Assume that no such state exists. Then, the outgoing edges for every state u with component $x_{\max}$ must terminate only in states with the component $x_{\max}$. Since the graph H

is assumed to be irreducible, this implies that the approximate eigenvector $\mathbf{x}$ is a constant vector, with all of its components equal to $x_{\max}$. If we divide both sides of the approximate eigenvector inequality ($A_H\mathbf{x} \geq n\mathbf{x}$) by $x_{\max}$, we see that the all-1's vector $\mathbf{1}$ is also an approximate eigenvector. However, this contradicts our assumption about A_H.

Let u be a state satisfying properties (5.3) and (5.4). We claim that $|E_u| \geq n$. To see this, observe that the approximate eigenvector inequality asserts

$$\sum_{v \in V_H} (A_H)_{u,v} x_v \geq n x_u \ .$$

Thus, by property (5.3) above,

$$|E_u|\, x_{\max} \geq \sum_{v \in V_H} (A_H)_{u,v} x_v \geq n x_u \ = \ n x_{\max} \ .$$

Dividing by $x_{\max}$ gives the desired conclusion $|E_u| \geq n$ (actually, with the help of property (5.4) above one can show that $|E_u| \geq n+1$, but this is not really needed now).

Let $M = |E_u| \geq n$. Write $E_u = \{e_1, e_2, \ldots, e_M\}$ and assume that e_1 terminates in state v (i.e., a state satisfying (5.4)), so $x_{\tau(e_1)} < x_{\max}$. Consider the partial accumulated weights

$$\theta_m = \sum_{i=1}^{m} x_{\tau(e_i)} \ , \quad m = 1, 2, \ldots, M$$

and their residues modulo n

$$\rho_m \equiv \theta_m \pmod{n} \ , \quad m = 1, 2, \ldots, M \ .$$

The *pigeon-hole principle*—which states that if one distributes n pigeons into n pigeon-holes, then either every hole has a pigeon, or some hole contains two or more pigeons—implies that the n residues, $\rho_1, \rho_2, \ldots, \rho_n$, satisfy one of the following conditions:

1. $\rho_m \equiv 0 \pmod{n}$ for some $1 \leq m \leq n$, or—
2. $\rho_{m_1} \equiv \rho_{m_2} \pmod{n}$ for some $1 \leq m_1 < m_2 \leq n$.

In the former case, we define a partition of E_u by setting

$$E_u^{(1)} = \{e_i\}_{i=1}^{m} \quad \text{and} \quad E_u^{(2)} = E_u - E_u^{(1)} \ .$$

In the latter case, we set

$$E_u^{(1)} = \{e_i\}_{i=m_1+1}^{m_2} \quad \text{and} \quad E_u^{(2)} = E_u - E_u^{(1)} \ .$$

In either case, the sum of weights of the edges in $E_u^{(1)}$ is divisible by n:

$$\sum_{e \in E_u^{(1)}} x_{\tau(e)} = rn \ .$$

Next we claim that

$$1 \le r < x_{\max} \ .$$

Clearly $1 \le r$ since $E_u^{(1)}$ is nonempty. To see that $r < x_{\max}$, observe that in the first case, $E_u^{(1)}$ contains at most n edges and includes e_1 for which $x_{\tau(e_1)} < x_{\max}$; and in the second case, $E_u^{(1)}$ has strictly fewer than n edges, each contributing at most $x_{\max}$ to the sum.

Now,

$$\begin{aligned} \sum_{e \in E_u^{(2)}} x_{\tau(e)} &= \sum_{e \in E_u} x_{\tau(e)} - \sum_{e \in E_u^{(1)}} x_{\tau(e)} \\ &\ge x_u n - rn \\ &= (x_u - r)n \ . \end{aligned}$$

Letting $y^{(1)} = r$ and $y^{(2)} = x_u - r$, we conclude that the partition,

$$E_u = E_u^{(1)} \cup E_u^{(2)} \ ,$$

defines a basic $\mathbf{x}$-consistent splitting. □

The discussion in this chapter implies a encoder construction procedure is known as the *state-splitting algorithm* or the *Adler-Coppersmith-Hassner (ACH) algorithm*. This algorithm is summarized in Figure 5.9.

Hints when constructing an encoder:

- In the course of a sequence of state splittings, the resulting graphs may become too unwieldy to draw. Instead, it may be more convenient to represent the graphs by tables; such a table has rows and columns indexed by the set of states with the (u, v)-entry containing the list of labels of all edges from u to v. Both the untagged encoder and the tagged encoder can also be represented in this way.

- If more then one round of splitting is required, it is convenient at each round to use the notation $u^{i,j}$ for each state. For instance, if state u has weight 5 and is split in the first round into two states, one of weight 3 and the other of weight 2, then after the first round, denote one of the descendant states by $u^{1,3}$ and the other by $u^{4,5}$. After the sequence of splittings is completed, the descendant states of u are denoted u^1, u^2, u^3, u^4, u^5.

1. Select a labeled graph G and integers p and q as follows:
   (a) Find a deterministic labeled graph G (or more generally a labeled graph with finite anticipation) which presents the given constrained system S.
   (b) Find the adjacency matrix A_G of G.
   (c) Compute the capacity $\mathsf{cap}(S) = \log \lambda(A_G)$.
   (d) Select a desired code rate $p : q$ satisfying
   $$\mathsf{cap}(S) \geq \frac{p}{q}$$
   (one usually wants to keep p and q relatively small for complexity reasons).
2. Construct G^q.
3. Using the Franaszek algorithm of Figure 5.3, find an $(A_G^q, 2^p)$-approximate eigenvector $\mathbf{x}$.
4. Eliminate all states u with $x_u = 0$ from G^q, and restrict to an irreducible sink H of the resulting graph. Restrict $\mathbf{x}$ to be indexed by the states of H.
5. Iterate steps 5a–5c below until the labeled graph H has minimum out-degree at least 2^p:
   (a) Find a non-trivial $\mathbf{x}$-consistent partition of the edges in H (the proof of Proposition 5.7 shows how to find such a partition in at least one state).
   (b) Find the $\mathbf{x}$-consistent splitting corresponding to this partition, creating a labeled graph H' and an approximate eigenvector $\mathbf{x}'$.
   (c) Let $H \leftarrow H'$ and $\mathbf{x} \leftarrow \mathbf{x}'$.
6. At each state of H, delete all but 2^p outgoing edges and tag the remaining edges with binary p-blocks, one for each outgoing edge. This gives a rate $p : q$ finite-state encoder for S.

Figure 5.9: State-splitting algorithm.

5.4 Strong decoders

We begin this section by proving Theorem 4.8, thereby showing how to achieve sliding-block decodability for finite-type constraints.

The proof is obtained by applying the state-splitting algorithm to any presentation of S with finite memory. Recall from Propositions 2.7 and 5.1 that higher powers and out-splitting preserve definiteness (although the anticipation may increase under out-splitting). Thus, the $(S^q, 2^p)$-encoder constructed in Section 5.3 is (m, a)-definite for some m and a and so, by Proposition 4.6, is sliding-block decodable. This completes the proof of Theorem 4.8.

Note that we can decode a q-block $\mathbf{w}$ as follows: observe the m previous q-blocks and the a upcoming q-blocks to determine the unique edge that produced $\mathbf{w}$; then read off the input tag on this edge. This defines an (m, a)-sliding-block decoder.

As an example, by examining Figure 5.8, one can see that the $(0,1)$-RLL encoder has a sliding-block decoder with window length 2, as shown in Table 4.1.

Next, we give a very simple example to illustrate how a tagged encoder can be non-catastrophic without being sliding-block decodable.

Consider the constrained system S which is presented by the labeled graph of Figure 5.10. Figure 5.11 exhibits a tagged $(S,3)$-encoder $\mathcal{E}$ which has finite anticipation; in fact it is deterministic. We claim that $\mathcal{E}$ is non-catastrophic. To see this, first observe that every symbol, except for d, is decoded consistently wherever it appears; so, the only decoding ambiguity is caused by the appearance of d as the label of two different edges with different input tags (namely, the edges labeled d outgoing from states 2 and 3). Now, suppose that two right-infinite sequences, $\mathbf{y}^+ = y_1y_2\cdots$ and $\mathbf{z}^+ = z_1z_2\cdots$, that can be generated by right-infinite paths in $\mathcal{E}$, differ in only finitely many places; then for some N and all $i > N$, we have $y_i = z_i$. We must show that when we decode $\mathbf{y}^+$ and $\mathbf{z}^+$ in a state-dependent manner, we get only finitely many differences in their decodings; more precisely, we must show that when $e_1e_2\cdots$ and $e'_1e'_2\cdots$ are right-infinite paths in $\mathcal{E}$ with output labels $\mathbf{y}^+$ and $\mathbf{z}^+$, then their input tags differ in only finitely many places. Well, e_i and e'_i will have the same input tag for $N < i < K$, where K is the first time after N that the symbol d appears in $\mathbf{y}^+$ (equivalently, $\mathbf{z}^+$), i.e., K is the smallest integer such that $K > N$ and $y_K = d$; if d does not appear at all after time N, then e_i and e'_i will have the same input tag for $i > N$, and we will be done. Now, at time K, we may decode $y_K = z_K = d$ in two different ways. But both occurrences of d in Figure 5.11 appear on edges with the same terminal state. So, for $i > K$, we will have $e_i = e'_i$, and decoding will be synchronized. Thus, the number of differences between the decodings of $\mathbf{y}^+$ and $\mathbf{z}^+$ will be at most $N+1$, and therefore our tagged encoder $\mathcal{E}$ is indeed non-catastrophic.

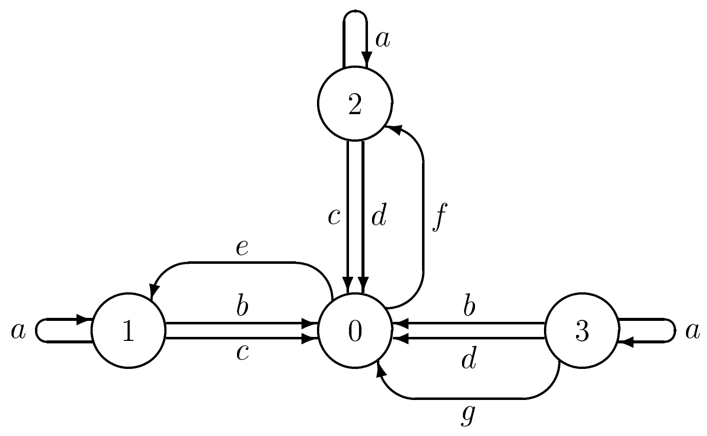

Figure 5.10: Graph presentation of constrained system S.

On the other hand, we claim that $\mathcal{E}$ is not sliding-block decodable. This follows immediately from the fact that for each ℓ, the symbol d in the word $a^\ell dea^\ell$ can appear on either

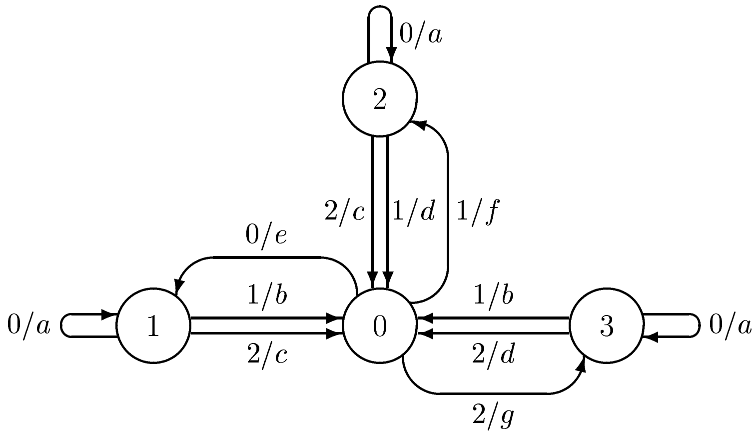

Figure 5.11: $(S, 3)$-encoder for constrained system S presented in Figure 5.10.

edge labeled d, yielding different decodings. In fact, for this particular constraint S, it turns out that there is no sliding-block decodable $(S, 3)$-encoder at all [KarM88].

We remark that if this code were to be used in conjunction with a noisy channel, then an isolated channel error would cause at most two 'bursts' of decoding errors: one burst from the channel error itself followed by a second burst caused by the ambiguity of decoding one occurrence of the symbol d. This 'two-burst' feature holds for the non-catastrophic encoders produced by the construction in Theorem 4.12.

We now give a very rough outline of the proof of Theorem 4.12; for details, see [KarM88]. Let S be an irreducible constrained system and let G be the Shannon cover of S. For simplicity, we assume that $q = 1$ (otherwise, by Theorem 3.7, we take an irreducible constrained system $S' \subseteq S^q$ with $\mathsf{cap}(S') = \mathsf{cap}(S)$). We must show that when $\log n \leq \mathsf{cap}(S)$, there is a non-catastrophic (S, n)-encoder $\mathcal{E}$, and if either $\log n < \mathsf{cap}(S)$ or S is almost-finite-type, then there is a sliding-block decodable (S, n)-encoder.

If $\log n < \mathsf{cap}(S)$, then it follows from Theorem 4.8 and Proposition 3.25 that there is a sliding-block decodable (S, n)-encoder. So, we may suppose that $\mathsf{cap}(S) = \log n$.

We need to introduce some terminology. A *generalized homing word* for a labeled graph is a word $\mathbf{w} = w_1 w_2 \ldots w_\ell$ such that for some $1 \leq r \leq \ell$, whenever $e_1 e_2 \ldots e_\ell$ and $e'_1 e'_2 \ldots e'_\ell$ are paths which generate $\mathbf{w}$, then $\tau(e_r) = \tau(e'_r)$; the integer r is called a *homing coordinate* for $\mathbf{w}$. Observe that in the special case that $r = \ell$, a generalized homing word is a homing word as in Section 2.6.3.

Our tagged encoder $\mathcal{E}$ will satisfy the following property:

(†) There is a generalized homing word $\mathbf{w} \in S(\mathcal{E}) \subseteq S$ for the encoder $\mathcal{E}$ and a positive integer M such that whenever a word $\mathbf{z} = z_{-M} z_{-M+1} \ldots z_M$ of length $2M+1$ in $S(\mathcal{E})$

does *not* contain $\mathbf{w}$, and $e_{-M}e_{-M+1}\ldots e_M$ and $e'_{-M}e'_{-M+1}\cdots e'_M$ are encoder paths which generate $\mathbf{z}$, then e_0 and e'_0 have the same input tag.

Such an encoder is bound to be non-catastrophic for roughly the same reason as the example above (in that example d is the (generalized) homing word): suppose that $\mathbf{y}^+ = y_1y_2\cdots$ and $\mathbf{z}^+ = z_1z_2\cdots$ are two right-infinite sequences that can be generated by the Shannon cover G such that $y_i = z_i$ for all i greater than some N; if we decode $\mathbf{y}^+$ and $\mathbf{z}^+$ then we will obtain the same decoded input tag sequence except possibly for two bounded bursts: a burst from time 0 to time $N{+}M$ and a burst from time $K{-}M$ to time $K{+}r$ where r is a homing coordinate for $\mathbf{w}$ and K is the first time after N at which $\mathbf{w}$ appears in $\mathbf{y}^+$; i.e., K is the smallest integer such that $K > N$ and $y_Ky_{K+1}\ldots y_{K+\ell(\mathbf{w})-1} = \mathbf{w}$; after time $K{+}r$, our decodings of $\mathbf{y}^+$ and $\mathbf{z}^+$ will be synchronized and we will get the same decoded symbol. Thus, our tagged encoder will indeed be non-catastrophic.

How do we construct a tagged encoder which satisfies (†)? Well, recall from Section 2.6 that there is a homing word $\mathbf{w}$ for the Shannon cover G. Let $S_{\mathbf{w}}$ denote the constrained system obtained from S by forbidding the appearance of the word $\mathbf{w}$. Since $\mathsf{cap}(S) = \log n$, it follows that $\mathsf{cap}(S_{\mathbf{w}}) < \log n$. From this, using a state splitting argument, starting with G, and deleting edges, (see [Mar85]), we construct a graph (V, E) endowed with two labelings—input labeling L_I and output labeling L_O—both with finite anticipation, such that

1. the labeled graph (V, E, L_O) presents $S_{\mathbf{w}}$;
2. the labeled graph (V, E, L_I) presents some proper constrained sub-system of the set of all n-ary words; and—
3. whenever γ and γ' are bi-infinite paths in (V, E) with the same L_O-labeling, they have the same L_I-labeling.

Then, by a long and delicate sequence of state splittings, both out-splitting and in-splitting, (see [KarM88]), the labeled graph (V, E, L_O) is transformed and extended to a presentation (V', E', L'_O) of all of S; moreover, this presentation has finite anticipation and constant out-degree n. At the same time, the labeling L_I is transformed to a deterministic labeling which serves as an input tagging L'_I of (V', E', L'_O). This gives our tagged (S, n)-encoder $\mathcal{E}$. While $\mathbf{w}$ need not be a generalized homing word for (V, E, L_O), it does determine a longer word which is a generalized homing word for (V', E', L'_O) and such that the condition (†) holds. This completes our outline of the construction of a non-catastrophic (S, n)-encoder $\mathcal{E}$.

It remains to show that if S is almost-finite-type, then $\mathcal{E}$ is actually sliding-block decodable. By definition, S is presented by some labeled graph with finite anticipation and finite co-anticipation (in fact, by Proposition 2.15, the Shannon cover G is such a presentation). Our encoder $\mathcal{E}$ is constructed from such a presentation by a sequence of state splittings, and thus it too has finite anticipation and finite co-anticipation. To show that it is sliding-block decodable, it suffices to show that whenever γ and γ' are bi-infinite paths in $\mathcal{E}$ with the same sequence, $\mathbf{y}$, of output labels, then they have the same sequence of input tags.

There are two cases to consider. If $\mathbf{y}$ contains $\mathbf{w}$—the generalized homing word for $\mathcal{E}$—then γ and γ' must arrive at the same state at some time, and so, by finite anticipation and finite co-anticipation, we must have $\gamma = \gamma'$; clearly then γ and γ' have the same sequence of input tags. Otherwise, $\mathbf{y}$ does not contain $\mathbf{w}$ and so by (†), γ and γ' again have the same sequence of input tags; thus, $\mathcal{E}$ is indeed sliding-block decodable, as desired.

5.5 Simplifications

5.5.1 State merging

In practice, it is desirable to design fixed-rate encoders with a small number of states. For a given labeled graph $G = (V, E, L)$, with an (A_G, n)-approximate eigenvector $\mathbf{x} = (x_v)_{v \in V}$, we have shown that the state-splitting algorithm can produce an encoder $\mathcal{E}$ with $|V_{\mathcal{E}}|$ states where

$$|V_{\mathcal{E}}| \leq \sum_{v \in V} x_v \ .$$

This gives an upper bound on the number of states in the smallest (S, n)-encoder. Often, however, one can reduce this number substantially by means of state merging. Although there is not yet a definitive solution to the problem of minimizing the number of encoder states, there are techniques and heuristics that have proved to be very effective in the construction of encoders.

One situation where we can merge states in a labeled graph H is the following. Let u and u' be two states in H and suppose that there is a 1–1 correspondence $e_i \mapsto e'_i$ between the sets of outgoing edges $E_u = \{e_1, e_2, \ldots, e_t\}$ and $E_{u'} = \{e'_1, e'_2, \ldots, e'_t\}$ such that for $i = 1, 2, \ldots, t$ we have $\tau(e_i) = \tau(e'_i)$ and $L(e_i) = L(e'_i)$. Then, we can eliminate one of the states, say u', and all of its outgoing edges, and redirect into u all incoming edges to u'. Clearly, the new labeled graph presents the same constraint, but with one fewer state. Note that this procedure is precisely the inverse of an in-splitting.

Example 5.4 Consider again the $(0, 1)$-RLL constrained system S. If we delete the self-loop $1 \xrightarrow{110} 1$ from Figure 5.5, we can see that states $0^{(2)}$ and 1 can be merged, according to the merging criterion just discussed. The resulting two-state labeled graph is the one shown in Figure 5.12. Tagging the latter, we obtain a tagged $(S^3, 2^2)$-encoder as shown in Figure 5.13; this is the same encoder presented in Figure 4.2. As mentioned in Example 4.2, this encoder is $(0, 1)$-definite. Hence, it is also $(0, 1)$-sliding-block decodable, and the respective decoding table is shown in Table 4.1. Note that the encoder of Figure 5.13 has fewer states than the encoder previously shown in Figure 5.8.

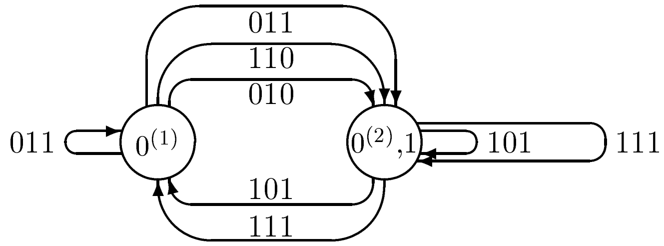

Figure 5.12: Deleting edges and merging states in Figure 5.5.

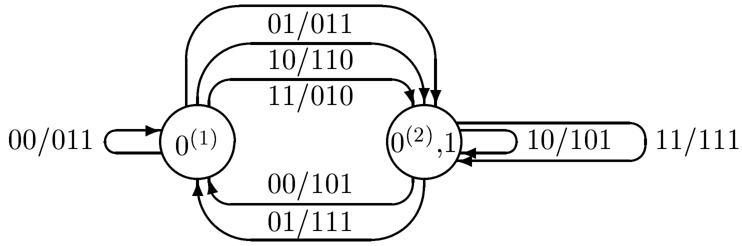

Figure 5.13: Rate 2 : 3 tagged two-state encoder for $(0,1)$-RLL constrained system.

Example 5.5 The second power of the Shannon cover of the $(1,3)$-RLL constrained system is given by the graph G in Figure 5.14. One can verify that

$$\mathbf{x} = (x_0\ x_1\ x_2\ x_3)^\top = (1\ 1\ 1\ 0)^\top$$

is an $(A_G^2, 2)$-approximate eigenvector. After deleting state 3 from G, we obtain an $(S(G),2)$-

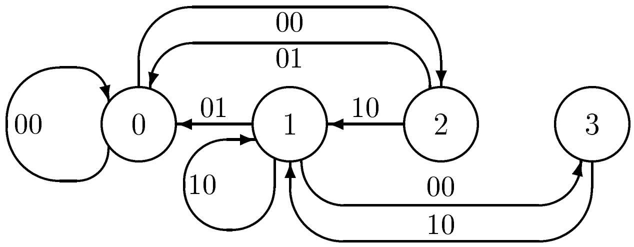

Figure 5.14: Second power of the Shannon cover of the $(1,3)$-RLL constrained system.

encoder in which states 1 and 2 are equivalent. When these two states are merged, we end up with the graph in Figure 5.15. This graph is an untagged version of the MFM encoder in Figure 4.1. $\square$

One key idea for merging states involves ordering the states in a labeled graph to reflect the inclusion relations among the follower sets at each state (defined in Section 2.6): given two states u and u' in a labeled graph G, we say that $u \preceq u'$ if the follower sets $\mathcal{F}_G(u)$

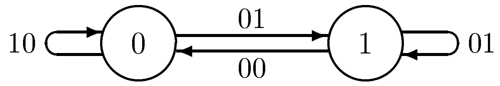

Figure 5.15: Untagged MFM encoder.

and $\mathcal{F}_G(u')$ satisfy $\mathcal{F}_G(u) \subseteq \mathcal{F}_G(u')$. This partial ordering of sets was used by Freiman and Wyner [FW64] in the construction of optimal block codes, mentioned in Section 4.2.

We now generalize the merging operation illustrated above as follows. Let G be a labeled graph and let u and u' be two states in G such that $u \preceq u'$. The (u, u')-*merger of* G is the labeled graph H obtained from G by:

1. eliminating all edges in $E_{u'}$;
2. redirecting into state u all remaining edges coming into state u';
3. eliminating the state u'.

Figures 5.16 and 5.17 show a schematic representation of a (u, u')-merger—before and after merging.

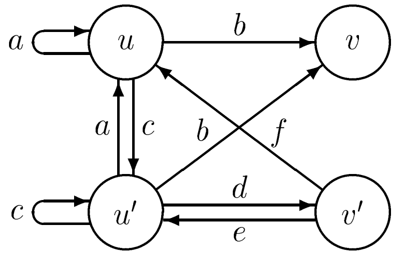

Figure 5.16: Local picture at states u and u' before merging.

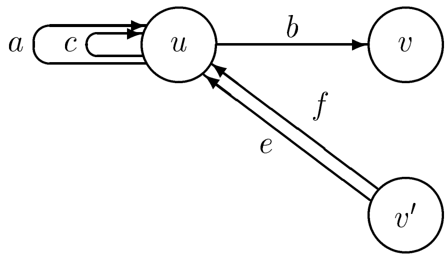

Figure 5.17: (u, u')-merger for Figure 5.16.

It is not hard to see that the merging of states performed on Figure 5.5 to obtain Figure 5.12 is a $(0^{(2)}, 1)$-merger.

The following result shows how we can reduce the final number of encoder states by (u, u')-merging.

Proposition 5.8 *Let G be a labeled graph, with an (A_G, n)-approximate eigenvector $\mathbf{x}$, and let u and u' be states in G satisfying*

(a) $u \preceq u'$, and —

(b) $x_u = x_{u'}$.

Let H denote the (u, u')-merger of G. Then

1. *$S(H) \subseteq S(G)$ and*
2. *The vector $\mathbf{y}$ defined by $y_v = x_v$ for all vertices v of H is an (A_H, n)-approximate eigenvector.*

Proof. 1. Let $\mathbf{w} = w_1 w_2 \ldots w_\ell$ be a word generated in H by a path $\gamma = e_1 e_2 \ldots e_\ell$. If γ does not contain any edge derived from an edge in G terminating in state u', one can immediately find a corresponding path $\hat{\gamma}$ in G that generates $\mathbf{w}$. Otherwise, let e_t be the last edge of γ that terminates in state u in H and comes from an edge $\hat{e}_t$ in G that terminates in state u'. By hypothesis (a) we have $\mathcal{F}_G(u) \subseteq \mathcal{F}_G(u')$, so there is a path $\hat{e}_{t+1}\hat{e}_{t+2}\ldots\hat{e}_\ell$ in G emanating from state u' and generating $w_{t+1}w_{t+2}\ldots w_\ell$. If $e_1 e_2 \ldots e_{t-1}$ contains no redirected edges, then $e_1 e_2 \ldots e_{t-1}\hat{e}_t\hat{e}_{t+1}\ldots\hat{e}_\ell$ is a path in G generating $\mathbf{w}$. If it does, let e_k be the last such edge. Then $e_{k+1}e_{k+2}\ldots e_{t-1}\hat{e}_t\hat{e}_{t+1}\ldots\hat{e}_\ell$ is a path in G that begins at state u and generates $w_{k+1}w_{k+2}\ldots w_\ell$. By hypothesis (a), there is another path $e'_{k+1}e'_{k+2}\ldots e'_\ell$ in G emanating from u' that also generates $w_{k+1}w_{k+2}\ldots w_\ell$. Continuing in this manner, we eventually produce a path in G that generates the entire word $\mathbf{w}$.

2. Let v be a state in H. By hypothesis (b),

$$(A_H\mathbf{y})_v = (A_G\mathbf{x})_v \geq n x_v = n y_v \ ,$$

so $\mathbf{y}$ is an (A_H, n)-approximate eigenvector, as desired. □

So, if states u and u' satisfy hypotheses (a) and (b) of the preceding result, then the number of final encoder states in the state-splitting construction is reduced by x_u.

In a set with partial ordering, there is the possibility of having minimal elements: a state u is *weight-minimal*, with respect to the partial ordering by follower sets and approximate eigenvector $\mathbf{x}$, if, for any other state v, the conditions $v \preceq u$ and $x_v = x_u$ imply that $u = v$.

Proposition 5.8 shows that, by means of preliminary state merging, encoder construction by the state-splitting algorithm can be accomplished using only the subgraph restricted to the weight-minimal states. This reduces the number of final encoder states to the sum of the weights of the weight-minimal states.

Example 5.6 Let G be some power of the Shannon cover of the (d,k)-RLL constrained system. Denoting the states by 0 through k (as in Figure 1.3), it is easy to verify that

$$i+1 \preceq i$$

for every $d \le i < k$.

Consider now the special case $(d,k) = (1,7)$ and let G be the third power of this constrained system. We have mentioned in Example 5.2 that

$$(2\ 3\ 3\ 3\ 2\ 2\ 2\ 0)^\top \tag{5.5}$$

is an $(A_G, 4)$-approximate eigenvector. With respect to this vector, the weight-minimal states in G are 0, 3, 6, and 7. We now delete state 7 (whose weight is zero) and merge the other states into the three remaining weight-minimal states as follows: states 1 and 2 are merged into state 3 to form state 1–3, and states 4 and 5 are merged into state 6 to form state 4–6. This, in turn, yields a graph G' with only three states, as shown in Figure 5.18. The

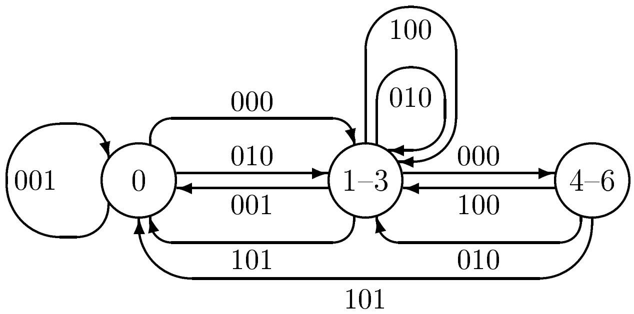

Figure 5.18: Merged graph G' for the $(1,7)$-RLL constrained system.

adjacency matrix of G' is given by

$$A_{G'} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 2 & 1 \\ 1 & 2 & 0 \end{pmatrix},$$

with an $(A_{G'}, 4)$-approximate eigenvector $(2\ 3\ 2)^\top$; in fact, this is a true eigenvector of $A_{G'}$ associated with the Perron eigenvalue 4.

If we now apply the state-splitting algorithm to the graph G', we can obtain a rate $2:3$ encoder for the $(1,7)$-RLL constrained system with at most $2+3+2=7$ states. This upper bound on the number of states is far better than the bound, 17, which we would obtain if the state-splitting algorithm were applied to the original presentation G with the approximate eigenvector in (5.5).

By applying further merging in the course of the state-splitting rounds of G', Weathers and Wolf obtained in [WW91] the encoder of Figure 4.6, which has only four states. It will follow from the discussion in Chapter 7 (Example 7.2) that no rate $2:3$ encoder for the $(1,7)$-RLL constrained system can have less than four states. □

The partial ordering on weight-minimal states also suggests certain out-splitting rounds and further state merging rounds than can simplify the final encoder graph. For instance, if $u \preceq v$ and $x_u < x_v$, it would be tempting to try to split state v into two states $v^{(1)}$ and $v^{(2)}$ with weights x_u and $x_v - x_u$ such that $v^{(1)}$ can be merged with state u. This would then further reduce the number of encoder states by x_u. In most cases of practical interest, this can be done. The paper [MSW92] describes one situation in general and several specific examples where the operations suggested by the partial ordering of the weight-minimal states can actually be implemented (those ideas were used by Weathers and Wolf in [WW91] to obtain their encoder in Figure 4.6). So, the merging principle is a valuable heuristic in encoder design.

However, as we show in Chapter 7, given a constrained system S and a positive integer n, there are lower bounds on the number of states that any (S,n)-encoder can have. In particular, there are limits on the amount of state merging that can be carried out.

Example 5.7 Let G be the 16th power of the $(2,10)$-RLL constrained system. One can verify through the Franaszek algorithm that

$$(1\;1\;1\;1\;1\;1\;1\;1\;1\;0\;0)^{\top}$$

is an $(A_G, 2^8)$-approximate eigenvector. The weight-minimal states in G are 0, 1, 8, and 10, and by deleting states 9 and 10 and merging states 2 through 7 into state 8 we obtain a labeled graph G' with three states, 0, 1, and 2–8. The adjacency matrix of G' is given by

$$A_{G'} = \begin{pmatrix} 83 & 57 & 117 \\ 122 & 83 & 170 \\ 1 & 85 & 173 \end{pmatrix} .$$

By further deleting edges we can obtain the 3-EFM(16) code (see Sections 1.7.2 and 4.4).

In Section 1.7.3, we showed how bit inversions in the output sequence of the encoder can reduce the DSV after precoding; such inversions effectively map certain input tags (bytes) into two possible codewords that differ in one bit. It was also mentioned, however, that the DSV reduction that can be obtained with the 3-EFM(16) code is not enough for optical applications.

The approach adapted in the DVD was designing an $(S(G), n)$-encoder where n is (significantly) greater than 256. The excess out-degree then allows to have $n-256$ input bytes each of which can be mapped into two different codewords; this flexibility in selecting the encoded codeword can then be used to reduce the DSV, especially if the two codeword candidates are such that one contains an even number of 1's while the other contains an odd number.

By Franaszek algorithm we obtain that there exists an (A_G, n)-approximate eigenvector whose largest component is 1 if and only if $n \le 260$; yet, an out-degree 260 is still too close to 256. Allowing the largest component in the approximate eigenvector to be 2, the values of n can go up to 351. For $n = 351$ we obtain the approximate eigenvector

$$(1\ 1\ 2\ 2\ 2\ 2\ 1\ 1\ 1\ 1\ 0)^\top ,$$

and the respective weight-minimal states in G are 0, 1, 5, 9, and 10. By deleting state 10 and merging states, we obtain a labeled graph H with four states, 0, 1, 2–5, and 6–9, whose adjacency matrix is given by

$$A_H = \begin{pmatrix} 83 & 57 & 98 & 22 \\ 122 & 83 & 142 & 32 \\ 164 & 113 & 192 & 42 \\ 97 & 67 & 113 & 25 \end{pmatrix}$$

and the respective $(A_H, 351)$-approximate eigenvector is

$$(1\ 1\ 2\ 1)^\top .$$

Next, we obtain a labeled graph H' by splitting state 2–5 into two descendant states, $2\text{–}5^{(1)}$ and $2\text{–}5^{(2)}$, with out-degrees 351 and 352. In fact, the splitting can be such that

$$\mathcal{F}_{H'}(2\text{–}5^{(1)}) \subseteq \mathcal{F}_{H'}(6\text{–}9) ,$$

thereby allowing merging. By deleting excess edges we thus obtain a four-state $(S(G), 351)$-encoder that is $(0, 1)$-sliding-block decodable. In particular, we can obtain in this manner the EFMPlus code, which is used in the DVD; see Section 1.7.3 and [Imm95b], [Imm99, Section 14.4.2]. □

5.5.2 Sliding-block decoder window

When a finite-state encoder with sliding block decoder is used in conjunction with a noisy channel, the extent of error propagation is controlled by the size of the decoder window. How large is this window? Well, suppose that we start the state-splitting algorithm with some labeled graph G presenting a constrained system of finite-type and G^q has finite memory $\mathcal{M} = \mathcal{M}(G^q)$ (measured in q-blocks). If t is the number of (rounds of) out-splitting used to construct the encoder, then the encoder graph $\mathcal{E}$ is $(\mathcal{M}, t)$-definite (measured in q-blocks). It follows that we can design a sliding-block decoder with decoding window length W satisfying the bound

$$\mathsf{W} \leq \mathcal{M} + t + 1$$

(again, measured in q-blocks). Recall from Section 5.3 that an upper bound on the number of (rounds of) out-splitting required is

$$t \leq \sum_{v \in V_G} (x_v - 1) ,$$

so,

$$\mathsf{W} \leq \mathcal{M} + \sum_{v \in V_G} (x_v - 1) + 1 . \tag{5.6}$$

The guarantee of a sliding-block decoder when S is finite-type and the explicit bound on the decoder window length represent key strengths of the state-splitting algorithm. In practice, however, the upper bound (5.6) on the window length often is larger—sometimes much larger—than the shortest possible.

For the $(0, 1)$-RLL encoder in Figure 5.8, where (referring to Figure 5.4) $\mathcal{M} = 1$, $x_0 = 2$, and $x_2 = 1$, this expression gives an upper bound of 3 (codewords) on the window length. However, we saw, in Table 4.1, a decoder with window length of $\mathsf{W} = 2$. For the rate $2 : 3$ $(1, 7)$-RLL encoder mentioned in Example 5.6 the initial labeled graph has memory $\mathcal{M} = 3$, and the approximate eigenvector is

$$(2\ 3\ 2)^\top .$$

The number of rounds of splitting turns out to be only 2, implying

$$\mathsf{W} \leq \mathcal{M} + 3 = 6 ,$$

which is, again, less than the upper bound (5.6) gives. In fact, a window length of $\mathsf{W} = 3$ was actually achieved [AHM82], [WW91]. For the rate $8 : 9$ $(0, G/I) = (0, 3/3)$ encoder for PRML discussed in [MSW92], the bound (5.6) was 11, but a window length of $\mathsf{W} = 2$ was achieved [MSW92].

These reduced window lengths were achieved by trying several possibilities for the choices of presentation, approximate eigenvector, out-splittings, elimination of excess edges and input tagging assignment (see, for instance [MSW92], [WW91]). In [KarM88] and [AM95], in-splitting was another tool used in reducing the window length—although for those codes, the ordinary state-splitting algorithm applied to a 'very large' approximate eigenvector will yield codes with the same sliding block decoding window. Recently, Hollmann [Holl95] has found an approach that combines aspects of the state-splitting algorithm with other approaches and has been demonstrated to be of use in further reducing the window length.

To illustrate the importance of the input tagging assignment, consider the encoders in Figures 5.19 and 5.20. In Figure 5.19 (which is the same as the MFM code of Figure 4.1), the decoder window length is one codeword, but in Figure 5.20, the minimum decoder window length is two codewords (one codeword look-back). The difference is that the assignment in the former is done in a more consistent manner: edges that have the same output label are assigned the same input tag.

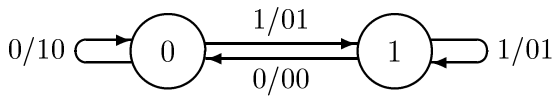

Figure 5.19: One choice of input tags.

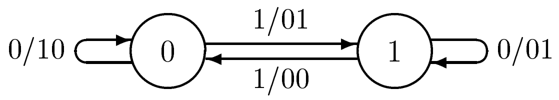

Figure 5.20: Another choice of input tags.

There is a 'brute-force' procedure for deciding if, given m and a, there is an input tagging assignment of a given (S, n)-encoder G which is (m, a)-sliding block decodable: for each edge e in G, let $L(e, \mathsf{m}, \mathsf{a})$ denote the set of all words generated by paths $e_{-\mathsf{m}} \ldots e_0 \ldots e_{\mathsf{a}}$ such that $e_0 = e$; it is straightforward to verify that an input tagging assignment on G is (m, a)-sliding block decodable if and only if whenever $L(e, \mathsf{m}, \mathsf{a})$ and $L(e', \mathsf{m}, \mathsf{a})$ intersect, e and e' have the same input tag.

In Figure 5.21, we exhibit an $(S, 2)$-encoder which is not block decodable (i.e., $(0, 0)$-sliding block decodable): the reader can easily verify that it is impossible to assign 1-bit input tags in such a way that the sliding-block decoder will have no look-back and no look-ahead. However, since the encoder is $(1, 0)$-definite, any input tag assignment allows a decoder window of length 2.

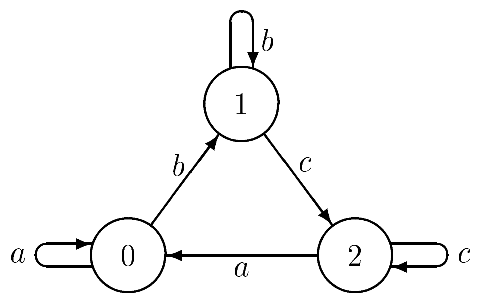

Figure 5.21: Encoder requiring decoder window length ≥ 2.

Theorem 4.9 shows that in general there is not much hope for simplifying the brute-force procedure, described above, when $n \geq 3$ (see the paragraph following the statement of Theorem 4.9).

5.6 Universality of the state-splitting algorithm

We end our treatment of the state-splitting algorithm by reviewing, without proof, some results from [AM95] and [RuR01] on the universality of the state-splitting algorithm.

Given a deterministic graph G, an integer n, and an (A_G, n)-approximate eigenvector $\mathbf{x}$, we say that the triple $(G, n, \mathbf{x})$ *splits into an* (S, n)*-encoder* $\mathcal{E}$ if there is sequence of rounds of out-splitting that can be applied to G such that the following holds:

1. The first round is consistent with $\mathbf{x}$, and each subsequent round is consistent with the respective induced approximate eigenvector.
2. The last round ends produces a labeled graph H in which each state has out-degree at least n.
3. The encoder $\mathcal{E}$ can be obtained from H by deleting excess edges and—in case $\mathcal{E}$ is tagged—by assigning input tags to the edges.

5.6.1 Universality for sliding-block decodable encoders

For integers m, a, and a function $\mathcal{D}$ from $(\mathsf{m}+\mathsf{a}+1)$-blocks of S to the n-ary alphabet (such as a sliding-block decoder), we define $\mathcal{D}_\infty^{\mathsf{m},\mathsf{a}}$ to be the induced mapping on bi-infinite sequences defined by:

$$\mathcal{D}_\infty^{\mathsf{m},\mathsf{a}}(\cdots w_{-1} w_0 w_1 \cdots) = \cdots s_{-1} s_0 s_1 \cdots \ ,$$

where

$$s_i = \mathcal{D}(w_{i-\mathsf{m}} \ldots w_{i-1} w_i w_{i+1} \ldots w_{i+\mathsf{a}}) \ .$$

Sometimes, we simply write $\mathcal{D}_\infty = \mathcal{D}_\infty^{\mathsf{m},\mathsf{a}}$. For a tagged (S, n)-encoder $\mathcal{E}$ with sliding-block decoder $\mathcal{D}$, we then have the mapping $\mathcal{D}_\infty$, and we take its domain to be the set of all bi-infinite (output) symbol sequences obtained from $\mathcal{E}$. When we refer to $\mathcal{D}_\infty$, we tacitly assume that its domain is included. We say that a mapping $\mathcal{D}_\infty$ is a *sliding-block* (S, n)*-decoder* if $\mathcal{D}$ is a sliding-block decoder for some tagged (S, n)-encoder.

We have the following positive results from [AM95].

Theorem 5.9 *Let S be an irreducible constrained system and let n be a positive integer.*

(a) Every sliding-block (S, n)-decoder has a unique minimal tagged (S, n)-encoder (here minimality can be taken to be in terms of number of states).

(b) If we allow an arbitrary choice of deterministic presentation G of S and (A_G, n)-approximate eigenvector $\mathbf{x}$, then the triple $(G, n, \mathbf{x})$ splits into a tagged (S, n)-encoder for every sliding-block (S, n)-decoder. If we also allow merging of states (i.e., (u, v)-merging

as in Section 5.5.1), then that triple splits into the minimal tagged (S, n)-encoder for every sliding-block (S, n)-decoder.

(c) If we fix G to be the Shannon cover S, but allow arbitrary (A_G, n)-approximate eigenvector $\mathbf{x}$, then $(G, n, \mathbf{x})$ splits into a tagged (S, n)-encoder for every sliding-block (S, n)-decoder $\mathcal{D}$, modulo a change in the domain of $\mathcal{D}_\infty$, possibly with a constant shift of each bi-infinite sequence prior to applying $\mathcal{D}_\infty$ (but with no change in the decoding function $\mathcal{D}$ itself). If we also allow merging of states, then, modulo the same changes, the triple $(G, n, \mathbf{x})$ splits into the minimal tagged (S, n)-encoder for every sliding-block (S, n)-decoder. In particular, the triple splits into a sliding-block (S, n)-decoder with minimal decoding window length.

On the other hand, we have the following negative results from [AM95].

1. If we fix G to be the Shannon cover of an irreducible constrained system S, then $(G, n, \mathbf{x})$ need not split into a sliding-block (S, n)-decoder with smallest number of encoder states in its minimal tagged (S, n)-encoder.

2. If we fix G to be the Shannon cover of an irreducible constrained system S and we fix $\mathbf{x}$ to be a minimal (A_G, n)-approximate eigenvector (in terms of the value of $\|\mathbf{x}\|_\infty$), then $(G, n, \mathbf{x})$ may fail to split into a sliding-block (S, n)-decoder with minimum decoding window length; examples of this kind were first found by Kamabe [Kam89] and Immink [Imm92], but in [AM95] an example is given where $\mathsf{cap}(S) = \log n$.

5.6.2 Universality for encoders with finite anticipation

Let u and u' be states in labeled graph G. We say that u and u' are 0-*strongly equivalent* if they are follower-set equivalent states; namely, $\mathcal{F}_G(u) = \mathcal{F}_G(u')$.

States u and u' in $G = (V, E, L)$ are t-*strongly equivalent* if the following conditions hold:

1. A one-to-one and onto mapping $\varphi : E_u \to E_{u'}$ can be defined from the set of outgoing edges of u to the set of outgoing edges of u', such that for every $e \in E_u$, both e and $\varphi(e)$ have the same label.

2. For every $e \in E_u$, the terminal states of e and $\varphi(e)$ are $(t-1)$-strongly equivalent.

States that are t-strongly equivalent are also r-strongly equivalent (and in particular they are equivalent states) for every $r < t$.

We say that two states are *strongly equivalent states* if for every $t \ge 0$ the states are t-strongly equivalent. So, when states are strongly equivalent, the infinite trees of paths that start in those states are the same. In a deterministic graph, two states are equivalent if and only if they are strongly equivalent. On the other hand, in a nondeterministic graph there may be two states that are equivalent but not strongly equivalent.

The following result is proved in [Ru96] and [RuR01].

Theorem 5.10 *Let S be an irreducible constraint and let n be a positive integer where $\mathsf{cap}(S) \geq \log n$. Suppose there exists some irreducible (S, n)-encoder $\mathcal{E}$ with $\mathcal{A}(\mathcal{E}) = t < \infty$. Then there exists an irreducible deterministic (not necessarily reduced) presentation G of S and an (A_G, n)-approximate eigenvector $\mathbf{x}$ that satisfy the following:*

(a) $\|\mathbf{x}\|_\infty \leq n^t$.

(b) The triple $(G, n, \mathbf{x})$ splits in t rounds into an (S, n)-encoder $\mathcal{E}_G$ such that $\mathcal{A}(\mathcal{E}_G) = t$.

(c) In each of the splitting rounds, every state is split into at most n states.

(d) In the ith round, the induced approximate eigenvector $\mathbf{x}^{(i)}$ satisfies $\|\mathbf{x}^{(i)}\|_\infty \leq n^{t-i}$.

(e) The encoder $\mathcal{E}$ can be obtained from $\mathcal{E}_G$ by merging strongly equivalent states.

Theorem 5.10 establishes the universality of the state-splitting algorithm for encoders with finite anticipation: every (S, n)-encoder with finite anticipation can be constructed using the state-splitting algorithm, combined with merging of (strongly) equivalent states, where the input to the process is some irreducible deterministic presentation G of S and an (A_G, n)-approximate eigenvector $\mathbf{x}$. (However, as shown in [RuR01], not always can the Shannon cover be taken as the graph G in Theorem 5.10.)

Another application of Theorem 5.10 is obtaining lower bounds on the anticipation of any (S, n)-encoder. We elaborate more on this in Section 7.4.3.

Problems

Problem 5.1 Construct the graph that is obtained by a complete out-splitting of the Shannon cover of the $(1, 3)$-RLL constrained system in Figure 1.16.

Problem 5.2 Let $\mathcal{E}_1$ be an (S, n)-encoder with finite anticipation. What can be said about the graph $\mathcal{E}_2$ that is obtained from $\mathcal{E}_1$ by complete out-splitting?

Problem 5.3 Let $G = (V, E, L)$ be an irreducible graph with $\lambda = \lambda(A_G)$ and let n be a positive integer such that $n < \lambda$. Denote by Δ the largest out-degree of any state in G.

Let $\mathbf{x} = (x_v)_{v \in V}$ be a strictly positive right eigenvector of A_G associated with the eigenvalue λ and let $x_{\min}$ the smallest entry in $\mathbf{x}$. Normalize the vector $\mathbf{x}$ so that $x_{\min} = \Delta/(\lambda - n)$, and define the integer vector $\mathbf{z} = (z_v)_{v \in V}$ by $z_v = \lfloor x_v \rfloor$, where $\lfloor y \rfloor$ stands for the largest integer not greater than y.

1. Show that
$$A_G \mathbf{z} \geq n\mathbf{z} > \mathbf{0}$$
(that is, $\mathbf{z}$ is an (A_G, n)-approximate eigenvector whose entries of are all positive).

2. Show that the sum of the entries in $\mathbf{z}$ satisfies the inequality
$$\sum_{v \in V} z_v \leq \frac{\Delta}{\lambda - n} \cdot \frac{\lambda^{|V|} - 1}{\lambda - 1} .$$

Problem 5.4 Let S be the $(0, \infty, 2)$-RLL constraint; that is, S consists of all binary words in which the runs of 0's in between consecutive 1's have even length. Denote by G the Shannon cover of S.

1. Construct the graphs G and G^3.
2. Are there any values m and a for which G^3 is (m, a)-definite?
3. Compute the base-2 capacity of S.
4. Construct a $(0, 1)$-definite $(S^3, 4)$-encoder with three states.
5. Does there exist an $(S^3, 4)$-encoder that is block decodable?

Problem 5.5 Let S be the constrained system presented by the graph G in Figure 4.11.

1. Construct an $(S^2, 2^3)$-encoder (i.e., a rate $3 : 2$ finite-state encoder for S) with two states.
2. Assign input tags to the encoder found in 1 so that it is block decodable (i.e., $(0, 0)$-sliding block decodable).

Problem 5.6 Let S be the constrained system presented by the graph G in Figure 5.22.

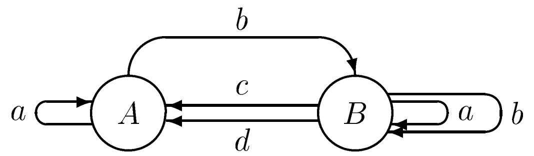

Figure 5.22: Graph G for Problem 5.6.

1. Compute the eigenvalues of A_G.

2. Construct a three-state (S, n)-encoder with $n = \lambda(A_G)$ by applying the state-splitting algorithm to G.
3. What is the anticipation of the encoder found in 2?
4. Let H be the graph in Figure 5.23. Is H an (S, n)-encoder?

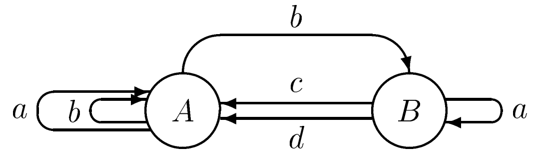

Figure 5.23: Graph H for Problem 5.6.

5. What is the anticipation of the graph H?
6. Can H be obtained by applying a sequence of rounds of state splitting to G (and merging equivalent states if there are any)? If yes, specify the sequence of state-splitting rounds; otherwise, explain.

Problem 5.7 Let S be the constrained system generated by the graph G in Figure 2.22.

1. Let $\mathcal{E}$ be an $(S, 3)$-encoder with the smallest number of states that can be obtained by an application of the state-splitting algorithm (without merging equivalent states) to the graph G. What is the number of states in $\mathcal{E}$?
2. Repeat 1 except that now $\mathcal{E}$ is an $(S^2, 2^3)$-encoder with the smallest number of states obtained by an application of the state-splitting algorithm (without merging equivalent states) on G^2.
3. Repeat 1 except that now $\mathcal{E}$ is an $(S^4, 2^6)$-encoder obtained from G^4.
4. What can be said about the anticipation of the encoder in 3?

Problem 5.8 Let G be the graph in Figure 5.24 in which the edges are assumed to have distinct labels.

1. Compute an $(A_G, 4)$-approximate eigenvector $\mathbf{x}$ in which the largest entry is the smallest possible.
2. Apply the state-splitting algorithm to G and the vector $\mathbf{x}$ from 1, and construct an $(S(G), 4)$-encoder using a minimal number of state-splitting rounds. What is the adjacency matrix of the resulting encoder?
3. Show that the encoder found in 2 satisfies the minimality criterion of number of rounds; that is, explain why no other $(S(G), 4)$-encoder can be obtained from $\mathbf{x}$ by less state-splitting rounds.

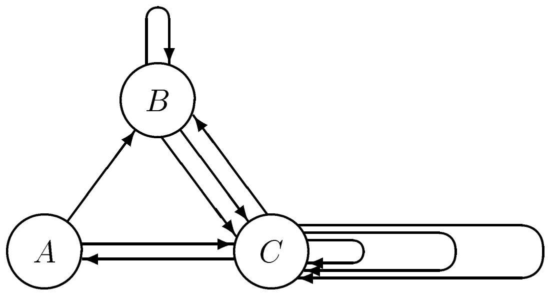

Figure 5.24: Graph G for Problem 5.8.

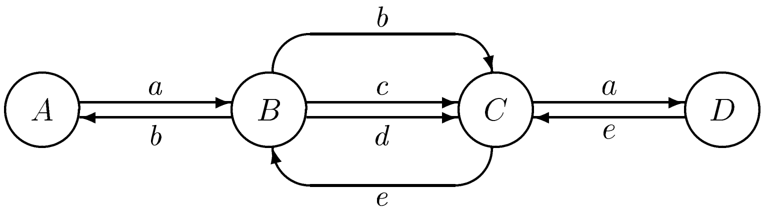

Figure 5.25: Graph G for Problem 5.9.

Problem 5.9 Let G be the graph in Figure 5.25 with labels over $\{a, b, c, d\}$.

1. What is the anticipation of G?

2. Compute the base-2 capacity of $S(G)$.

3. Compute an $(A_G, 2)$-approximate eigenvector in which the largest entry is the smallest possible.

4. Apply the state-splitting algorithm to G and the vector found in 3, and construct an $(S(G), 2)$-encoder with anticipation which is the smallest possible.

5. The anticipation of the encoder found in 4 is smaller than the anticipation of G. Explain how this could happen, in spite of obtaining the encoder by splitting states in G.

Problem 5.10 Let G be a deterministic graph and let $\mathbf{x}$ be an (A_G, n)-approximate eigenvector. Further, assume that an $(S(G), n)$-encoder $\mathcal{E}$ can be obtained from G by *one* $\mathbf{x}$-consistent round of state splitting (and deleting redundant edges), and the resulting $(A_{\mathcal{E}}, n)$-approximate eigenvector is a 0–1 vector.

1. Show that $\mathbf{x}$ must satisfy the inequality

$$A_G \mathbf{1} \geq \mathbf{x} ,$$

where $\mathbf{1}$ is the all-one vector.

2. Suppose that, in addition, $\mathbf{x}$ does *not* satisfy the inequality

$$A_G \mathbf{1} \geq 2\mathbf{x} \ .$$

Show that one of the entries in $\mathbf{x}$ is at least n.

Hint: Show that there is a state u in G such that every $\mathbf{x}$-consistent partition of the outgoing edges from u must contain a partition that consists of exactly one edge.

Problem 5.11 Let S be the constrained system presented by the graph G in Figure 2.23.

1. Construct an $(S^4, 2^3)$-encoder (i.e., a rate $3:4$ finite-state encoder for S) with two states.
2. Is it possible to assign input tags to the encoder found in 1 so that it is block decodable (i.e., $(0,0)$-sliding block decodable)? If yes, suggest such a tag assignment; otherwise, explain.

Problem 5.12 Let S be the constrained system generated by the graph G in Figure 5.26.

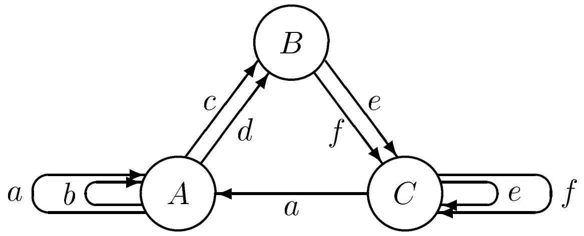

Figure 5.26: Graph G for Problem 5.12.

1. Find the smallest value of M for which there exists an $(A_G, 3)$-approximate eigenvector whose largest entry equals M.
2. Among all $(A_G, 3)$-approximate eigenvectors whose largest entries equal the value M found in 1, find a vector $\mathbf{x}$ whose sum of entries is minimal.
3. Construct an $(S, 3)$-encoder $\mathcal{E}$ by applying the state-splitting algorithm to G and the vector $\mathbf{x}$ found in 2.
4. How many state-splitting rounds were required in 3? Why can't an $(S, 3)$-encoder be obtained from G and $\mathbf{x}$ using a smaller number of rounds?
5. What is the anticipation of $\mathcal{E}$?
6. An (m, a)-sliding block decoder is sought for the encoder $\mathcal{E}$. Find the smallest possible values of m and a for which such a decoder exists regardless of the tag assignment to the edges in $\mathcal{E}$.

7. Is it possible to have an (m, a)-sliding block decoder for $\mathcal{E}$ with a smaller m by a clever tag assignment to the edges in $\mathcal{E}$? If yes, suggest such a tag assignment. Otherwise explain.

8. Repeat 7 with respect to the parameter a.

Problem 5.13 Let S be the constrained system presented by the graph G in Figure 5.27 (which is the same as Figure 2.25).

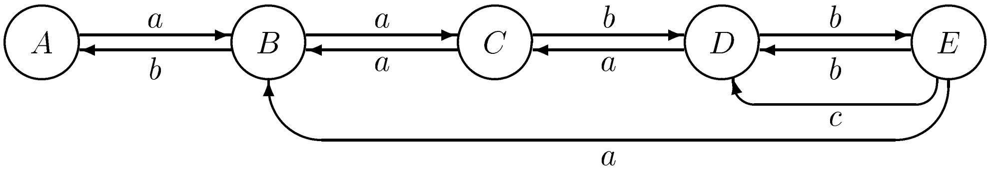

Figure 5.27: Graph G for Problem 5.13.

1. Compute the base-2 capacity of S.

2. Compute an $(A_G, 2)$-approximate eigenvector in which the largest entry is the smallest possible.

3. Show that an $(S, 2)$-encoder can be obtained by four rounds of splitting of G.

4. Construct a $(0, 1)$-definite $(S^2, 4)$-encoder with two states.

Problem 5.14 Let S be the constrained system presented by the graph G in Figure 2.24.

1. Compute the base-2 capacity of S.

2. Using the state-splitting algorithm, construct an $(S^2, 2)$-encoder (i.e., a rate $1:2$ finite-state encoder for S) with six states and anticipation at most 1.

3. Is it possible to assign input tags to the encoder in 2 so that it is block decodable? If yes, suggest such an assignment; otherwise, explain.

4. Show how to construct an $(S^2, 2)$-encoder with finite anticipation yet that anticipation is greater than 1.

5. Is there a positive integer ℓ for which there exists a block $(S^{2\ell}, 2^\ell)$-encoder (i.e., a rate $\ell : 2\ell$ one-state encoder for S)? If yes, construct such an encoder; otherwise, explain.

Problem 5.15 Let S be the constrained system generated by the graph G in Figure 5.28.

1. What is the memory of G?

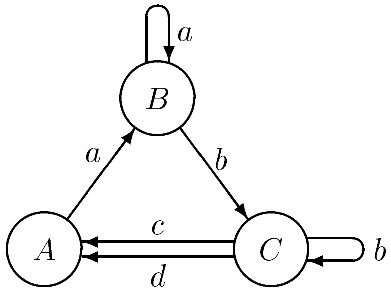

Figure 5.28: Graph G for Problem 5.15.

2. Compute the base-2 capacity of $S(G)$.
3. Compute an $(A_G, 2)$-approximate eigenvector $\mathbf{x}$ in which the largest entry is the smallest possible.
4. Apply the state-splitting algorithm to G and the vector $\mathbf{x}$ from 3, and construct an $(S(G), 2)$-encoder using a minimal number of state-splitting rounds. Merge states to obtain an encoder with three states. What is the anticipation of this encoder?
5. Assign input tags to the encoder found in 4 so that it is (m, a)-sliding-block decodable with the smallest possible window length $\mathsf{m} + \mathsf{a} + 1$.
6. Is there a positive integer ℓ for which there exists a deterministic $(S^\ell, 2^\ell)$-encoder? If yes, construct such an encoder; otherwise, explain.

Problem 5.16 Let G be the second power of the Shannon cover of the $(2, 7)$-RLL constrained system and let $bldx$ be the vector

$$(2\ 3\ 4\ 4\ 3\ 3\ 1\ 1)^\top .$$

As mentioned in Example 5.3, this vector is an $(A_G, 2)$-approximate eigenvector.

1. Find the weight-minimal states in G with respect to $\mathbf{x}$.
2. By merging states in G into its weight-minimal states, construct a deterministic graph G' with five states such that $S(G') \subseteq S(G)$ and $\mathsf{cap}(S(G')) \geq 1$.