Chapter 6

Other Code Construction Methods

In Chapter 5, we focused on the state-splitting algorithm which gives a rigorous proof of the general existence theorems (Theorems 4.1 and 4.8) for constrained code constructions. The algorithm has other virtues: it is fairly easy to use, it has yielded many codes of practical interest, and, as stated in Section 5.6, it is a universal method of constructing every sliding-block decoder. However, as pointed out in Section 5.5, there are lots of choices in the algorithm: the presentation, the approximate eigenvector, the sequence of out-splittings, the deletion of excess edges, and the input-tagging assignment. It is not known how to make these choices to yield optimal encoders/decoders.

There have been many other important and interesting approaches to constrained code construction—far too many to mention here. In this chapter we review, without proof, some alternative methods for constructing constrained codes (see also Sections 4.2 and 4.4). Since these methods are all aimed at the same goal, it is perhaps not surprising that they have a lot in common. In fact, other methods are probably just as universal as the state-splitting algorithm. Some of these methods give very useful heuristics for constructing sliding-block decodable encoders with small decoding window length as well as some rigorous upper bounds on the smallest possible anticipation (or decoding delay) and decoding window length.

We will focus on constructions of tagged (S, n)-encoders. So, throughout this chapter, we will always assume that S is an irreducible constrained system with $\mathsf{cap}(S) \geq \log n$. For rate $p : q$ finite-state encoders for S, one can apply the results and methods of this chapter by passing to the power S^q.

6.1 IP encoders

We start with an encoder construction due to Franaszek that is very closely related to the state-splitting construction. We need the following notation: for a path γ in a labeled graph

$G = (V, E, L)$, let $\hat{\gamma}$ denote the truncation of γ obtained by deleting the last edge, and for a set Γ of paths in G, let $\hat{\Gamma}$ denote the set $\{\hat{\gamma} : \gamma \in \Gamma\}$.

Now, for each state $u \in V$, let $N = N(u)$ be a positive integer and let $\Gamma_u = \{\Gamma_u^{(1)}, \Gamma_u^{(2)}, \ldots, \Gamma_u^{(N)}\}$ be a partition of a set of paths outgoing from u, of some fixed length T, subject to the following condition:

IP condition: Whenever e is an edge from state u to state v and γ_1 and γ_2 are paths that belong to the same element of Γ_v (in particular, they both have initial state v), then $e\hat{\gamma}_1$ and $e\hat{\gamma}_2$ belong to the same element of Γ_u.

Note that this is a condition imposed on the entire collection of partitions, $\{\Gamma_u\}_{u \in V}$, not on an individual partition Γ_u. The IP condition was developed by Franaszek [Fra80a], [Fra80b], [Fra82], who called the individual partition elements, *independent path sets* or *IP sets*. The IP condition was called the *left Markov property* in [AM95].

Now suppose that $\{\Gamma_u\}_{u \in V}$ is a collection of partitions satisfying the IP condition. Construct the following labeled graph $G' = (V', E', L')$: the states in G' are the independent path sets $\Gamma_u^{(i)}$; there is an edge $\Gamma_u^{(i)} \xrightarrow{b} \Gamma_v^{(j)}$ in G' for each edge e labeled b from state u to state v in G such that

$$e\,\widehat{\Gamma}_v^{(j)} \subseteq \Gamma_u^{(i)} .$$

Now, it can be shown that G' is lossless (Problem 6.1). Moreover, if the vector $\mathbf{x}$ defined by $x_u = N(u)$ is an (A_G, n)-approximate eigenvector, then G' has minimum out-degree at least n; so, by deleting edges and adding input tags, we obtain an ordinary tagged (S, n)-encoder, which we call an *IP encoder*. If G has finite memory $\mathcal{M}$, then this encoder is sliding-block decodable with decoding window length $\leq \mathcal{M}+T+1$.

It turns out that any IP encoder can also be constructed by $\mathbf{x}$-consistent out-splittings (see [AM95] and [Holl95]). It follows from this result and the state-splitting construction in Chapter 5 that the IP approach is bound to succeed provided that T is taken sufficiently large. In fact, one can view the state-splitting algorithm as giving a constructive procedure for manufacturing IP sets. Recently, Franaszek and Thomas [FT93] discovered an algorithm which reduces the search required for finding IP sets.

6.2 Stethering encoders

Next, we describe a more general framework of encoder construction within which the state-splitting construction fits. Let $G = (V, E, L)$ be a deterministic labeled graph with $S = S(G)$ and let $\mathbf{x} = (x_u)_{u \in V}$ be an (A_G, n)-approximate eigenvector. As usual, we may assume that the entries of $\mathbf{x}$ are all strictly positive. We describe a particular class of (S,n)-encoders $\mathcal{E}$, built from $\mathbf{x}$, as follows. The set of states of $\mathcal{E}$ is given by

$$V_{\mathcal{E}} = \left\{ (u,i) \;\middle|\; u \in V_G \quad \text{and} \quad i = 0,1,\ldots,x_u{-}1 \right\} .$$

Recall that E_u denotes the set of edges outgoing from a state u, and we will use $L(E_u)$ here to denote the set of L-labels of edges in E_u. Since G is deterministic, the mapping $E_u \to L(E_u)$ defined by $e \mapsto L(e)$ is one-to-one and onto. So, it makes sense, for each $u \in V_G$ and $a \in L(E_u)$, to define $\tau(u;a)$ to be the terminal state of the unique edge outgoing from u with label a. Now, let

$$\Delta_u = \{(a,j) \,:\, a \in L(E_u) \quad \text{and} \quad j = 0,1,\ldots,x_{\tau(u;a)}{-}1\} .$$

By definition of approximate eigenvector we have $|\Delta_u| \ge n x_u$. Thus, we can partition Δ_u into $x_u{+}1$ subsets $\Delta_u^{(0)}, \Delta_u^{(1)}, \ldots, \Delta_u^{(x_u)}$ such that $|\Delta_u^{(i)}| = n$ for each i, except for $i = x_u$ ($\Delta_u^{(x_u)}$ may be empty). Now, for each $i = 0,1,\ldots,x_u{-}1$ and $(a,j) \in \Delta_u^{(i)}$, we endow $\mathcal{E}$ with an edge $(u,i) \xrightarrow{a} (\tau(u;a),j)$. This completely defines $\mathcal{E}$.

It turns out that $\mathcal{E}$ is an (S,n)-encoder (the proof is essentially contained in [AGW77]—see also [AMR95]); in fact, if G were merely lossless, then one could modify the construction so that $\mathcal{E}$ would still be an (S,n)-encoder. Also, it is not hard to see that the (S,n)-encoders constructed by the state-splitting algorithm are of this type. Clearly the construction of this type of encoder is somewhat easier than the state-splitting construction: there is no iterative process to go through. However, there are lots of examples of encoders of this type that do not have finite anticipation; it is not at all clear how to choose the partitions of each Δ_u to achieve finite anticipation.

On the other hand, if we have sufficient excess capacity and we choose our partitions of Δ_u with some extra care, then it turns out that $\mathcal{E}$ will have finite anticipation; and if G has finite memory (more generally, if G is definite), then $\mathcal{E}$ will be definite, and so any tagging of $\mathcal{E}$ will yield a sliding-block decodable encoder. We describe this special construction as follows.

The definition of the partitions assumes some ordering on the edges in E_u for each $u \in V_G$—equivalently, in light of the assumption that G is deterministic, an ordering on the symbols in $L(E_u)$. We allow symbols to have different ordering relations in $L(E_u)$ for different states u. For $u \in V_G$ and $a \in L(E_u)$, define $\phi(u;a)$ by

$$\phi(u;a) = \sum_{\{b \in L(E_u)\,:\, b<a\}} x_{\tau(u;b)} ,$$

where the sum is zero on an empty set. For $i = 0,1,\ldots,x_u{-}1$, let

$$\Delta_u^{(i)} = \{\, (a,j) \,:\, a \in L(E_u) \quad \text{and} \quad in \le \phi(u;a) + j < (i+1)n \,\} ,$$

and $\Delta_u^{(x_u)}$ is whatever is left over. This means that for each $u, v \in V_G$, $0 \le i < x_u$, $0 \le j < x_v$, and symbol a, we have one edge $(u,i) \xrightarrow{a} (v,j)$ in $\mathcal{E}$ if and only if

$$a \in L(E_u) , \quad v = \tau(u;a) , \quad \text{and} \quad in \le \phi(u;a) + j < (i+1)n . \tag{6.1}$$

Such an encoder is called a *stethering* (S,n)*-encoder* because the elements of each partition element $\Delta_u^{(i)}$ form a contiguous interval and thus are 'stuck together.' This construction is illustrated in Figure 6.1: there is an edge from each state (u,i) in the first row to each state in the second row that sits below (u,i).

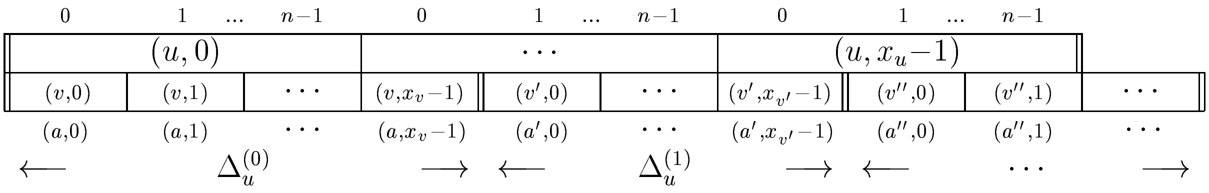


Figure 6.1: Stethering encoders.

Suppose that we have enough excess capacity that $\mathsf{cap}(S) \geq \log(n+1)$. Then, using an $(A_G, n+1)$-approximate eigenvector, we can form a stethering $(S, n+1)$-encoder $\mathcal{E}$. Now, for each state u, the ordering on $L(E_u)$ induces a lexicographic ordering on Δ_u, and hence an ordering on each $\Delta_u^{(i)}$. We tag $\mathcal{E}$ by assigning the input tags $\{0,1,\ldots,n\}$ in an ordering-preserving way to the outgoing edges specified by each $\Delta_u^{(i)}$. From $\mathcal{E}$ we form an (S,n)-encoder $\mathcal{E}'$ by deleting all edges in $\mathcal{E}$ tagged by the input symbol n. We call such an encoder a *punctured stethering* (S,n)*-encoder*. In [AMR95], it is shown that any punctured stethering (S,n)-encoder has finite anticipation.

Now, what is the advantage of such an encoder? Well, not only is it easy to construct, but, as we will discuss in Section 7.4, its anticipation is in some sense 'small,' especially compared to the upper bound on anticipation that we gave in Section 5.3. Also, with sufficient excess capacity, this construction leads to sliding-block decoders with 'small' decoding window length (see the discussion in Section 7.4).

6.3 Generalized tagged (S,n)-encoders

In order to describe some of the other approaches to code construction, we will now formulate what appears to be a more general notion of tagged (S,n)-encoder. However, these more general encoders can be transformed to ordinary tagged (S,n)-encoders.

A *generalized tagged* (S,n)*-encoder* $\mathcal{E} = (V, E, L_I/L_O)$ is a finite directed graph (V,E) endowed with two labelings—input labeling (tagging) L_I and output labeling L_O—yielding two labeled graphs $G_I = (V, E, L_I)$ and $G_O = (V, E, L_O)$ such that

1. G_I presents the unconstrained system of all n-ary sequences;

2. G_I has finite anticipation;
3. $S(G_O) \subseteq S$;
4. G_O is lossless.

As with ordinary tagged encoders, we will use the notation $u \stackrel{s/b}{\to} v$ for an edge from state u to state v in $\mathcal{E}$ with L_I-labeling (tagging) s and L_O-labeling b.

The main difference between a generalized tagged encoder and an ordinary tagged encoder is that the input tagging is merely required to have finite anticipation, rather than to be deterministic. However, since G_I presents all n-ary sequences, it can be shown directly that $(V, E, L_I/L_O)$ can encode unconstrained data into S at the cost of finite delay. Alternatively, we can transform any generalized tagged (S, n)-encoder into an (ordinary) tagged (S, n)-encoder. This fact is implicit in many papers (e.g., [AFKM86], [Heeg91, p. 770]) and has recently been made more explicit by Hollmann [Holl95]. However, it is also a simple consequence of an old symbolic dynamics result. The transformation is outlined in Section 6.7.

6.4 Encoders through variable-length graphs

6.4.1 Variable-length graphs and n-codability

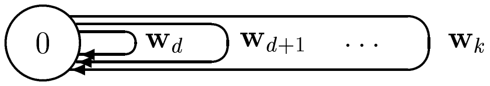
In the following, we show how a certain kind of labeled variable-length graph yields a generalized tagged (S, n)-encoder (and therefore an ordinary tagged (S, n)-encoder).

A *(labeled) variable-length graph* (in short, VLG) is a labeled finite directed graph $G = (V, E, L)$ in which each edge e has a length $\ell(e)$, assumed to be a positive integer, and is labeled by a word $L(e)$ (in some alphabet) of length $\ell(e)$. An ordinary labeled finite directed graph is a VLG where each edge has length 1.

Any VLG, G, may be viewed as an ordinary labeled graph by inserting dummy states. In this way, a VLG presents a constrained system. So, the notation $S(G)$ makes sense for a VLG, and we can apply the various notions of determinism, losslessness, definiteness, finite memory, finite anticipation, etc., to VLG's. An advantage of VLG's is that they tend to be more compact representations of constraints. For instance, any (d, k)-RLL constraint naturally has an ordinary presentation with $k+1$ states and a VLG presentation with only one state, as shown in Figure 6.2.

A VLG G is *n-codable* if at each state $u \in V_G$, the length distribution of the set E_u of outgoing edges satisfies the Kraft inequality with equality, namely, we have

$$\sum_{e \in E_u} n^{-\ell(e)} = 1 .$$



$$\mathbf{w}_i = \underbrace{00\ldots0}_{i}1$$

Figure 6.2: VLG presentation of (d, k)-RLL constrained system.

Now, suppose that G is an n-codable lossless VLG presentation of a subset of a constrained system S. Then, we can select, for each state $u \in V_G$, a prefix-free list X_u of n-ary words whose length distribution matches that of E_u. Using any length-preserving correspondence between X_u and E_u, we endow G with an input tagging L_I. It can be shown that $(V, E, L_I/L)$, viewed as an ordinary graph with two labelings—input tagging L_I and output labeling L—is a generalized tagged (S, n)-encoder (Problem 6.2).

6.4.2 Variable-length state splitting

Variable-length state splitting is a code construction technique, due to Adler, Friedman, Kitchens, and Marcus [AFKM86] (see also Heegard, Marcus, and Siegel [Heeg91]), which adapts the state-splitting algorithm to variable-length graphs. In this method, we begin with an ordinary lossless presentation $G = (V, E, L)$ of a constrained system S, and we select a subset $V' \subseteq V$ of states such that every sufficiently long path in G meets some state of V'. We then convert G into a VLG $G' = (V', E', L')$, where E' is the set of paths in G which start and terminate in V', but do not contain any states of V' in their interior, with the naturally inherited labeling (the length of an edge in G' is, of course, the length of the path in G that it represents). The assumption on V' guarantees that every bi-infinite path in G can be parsed into edges of G', and so we do not really lose anything in passing from G to G'. We remark that if we view G' as an ordinary labeled graph (by inserting dummy states), then, while G' may not exactly reproduce G, it is a lossless labeled graph with $S(G') = S(G)$.

One can develop notions of state splitting, approximate eigenvectors, etc. for VLG's, and see that state splitting preserves finite anticipation and definiteness as in Proposition 5.1. Then we apply a variable-length version of the state-splitting algorithm to iteratively transform G' into a new VLG which is n-codable, thereby yielding a generalized tagged (S, n)-encoder. If the original presentation G has finite anticipation (resp., has finite memory), then the encoder will have finite anticipation (resp., be sliding-block decodable).

The variable-length state-splitting procedure offers the advantage of a simpler construction method because there are fewer states to split. In some cases, it also suggests ways to find codes with reduced complexity (see [AFKM86], [Heeg91] for examples).

6.4.3 Method of poles

Our next alternative method is the *method of poles*, due to Béal [Béal90a], [Béal93a], [Béal90b]. This method is a modification of the *method of principal states*, developed by Franaszek [Fra69]. A related method was used in [HorM76] to obtain the first practical rate $2:3$ finite-state encoder for the $(1,7)$-RLL constraint.

In this method, we assume that $\mathsf{cap}(S) > \log n$, although it does work sometimes even when $\mathsf{cap}(S) = \log n$ (see [AB94]). Again, we begin with an ordinary lossless presentation G of S. Given a positive integer M, we find a subset P of V, called *principal states*, and, for each $u \in P$, a collection $\Gamma(u)$ of paths in G satisfying the following four conditions:

(PS-1) each path in $\Gamma(u)$ starts at u;

(PS-2) each path terminates in some element of P;

(PS-3) each path has length at most M;

(PS-4) the reverse Kraft inequality holds at every state $u \in P$, namely,

$$\sum_{\gamma \in \Gamma(u)} n^{-\ell(\gamma)} \geq 1 \ .$$

Note that, in contrast to variable-length state splitting, paths in $\Gamma(u)$ are allowed to visit states of P in their interior (i.e., not just at the beginning or end). In Section 4.4 we presented a special case of this definition, where each $\Gamma(u)$ consisted of paths of length exactly $q = M$.

Assuming $\mathsf{cap}(S) > \log n$, for sufficiently large M, such a set of principal states always exists [Béal90a]. There are algorithms which give short-cuts to an exhaustive search for sets of principal states [Fra69], [Béal90a].

A generalized tagged (S,n)-encoder with sliding-block decoder can be found using the lists $\Gamma(u)$ as follows. We first extract a subset $\Gamma'(u) \subseteq \Gamma(u)$ such that the following condition holds:

(PS-4') $\sum_{\gamma \in \Gamma'(u)} n^{-\ell(\gamma)} = 1.$

This can always be done [Béal90a]. We now define a VLG presentation $\overline{G} = (\overline{V}, \overline{E}, \overline{L})$ of a subset of $S = S(G)$ as follows. For each principal state $u \in P$, we endow $\overline{G}$ with a state $\overline{u}$, called a *pole state*, and for each path $\gamma \in \Gamma'(u)$ in G terminating in a state $v \in P$, we endow $\overline{G}$ with an edge $\overline{\gamma} = \overline{u} \to \overline{v}$ of length $\ell(\gamma)$. The edge $\overline{\gamma}$ in $\overline{G}$ is labeled with the word generated by the corresponding path γ in G.

Clearly, $S(\overline{G}) \subseteq S(G) = S$, and it is not hard to see that since G is lossless, so is $\overline{G}$. Moreover, the VLG $\overline{G}$ is n-codable. So, as described in Section 6.4.1, we can endow $\overline{G}$ with an input tagging to obtain a generalized tagged (S, n)-encoder.

Now, the presentation $\overline{G}$ need not be definite even if G has finite memory. However, Béal [Béal90a] showed that if G does have finite memory and the lists $\Gamma'(u)$ satisfy an optimization condition (condition (PS-5) below), then $\overline{G}$ will be definite. Namely, she showed that if G has finite memory and $P \subseteq V_G$ is a set of states such that for each $u \in P$, the list $\Gamma'(u)$ satisfies conditions (PS-1)–(PS-3), (PS-4'), and (PS-5), then $\overline{G}$ is definite, and so any tagging of the resulting (S, n)-encoder will be sliding-block decodable. The additional condition (PS-5) is as follows:

(PS-5) for each state $u \in P$, the list $\Gamma'(u)$ minimizes the sum $\sum_{\gamma \in \Gamma'(u)} \ell(\gamma)$, among all possible lists satisfying conditions (PS-1)–(PS-3) and (PS-4').

We note that Béal [Béal93b] has found other optimality conditions that can replace condition (PS-5) for guaranteeing sliding-block decodability.

6.5 Look-ahead encoders

Next, we discuss some methods, due to Franaszek [Fra79], [Fra82], [Fra89], which predate the state-splitting approach, but involve some very closely related ideas. In our description, we adopt the viewpoint of Hollmann [Holl95], whose terminology is somewhat different from that of Franaszek.

We begin with look-ahead encoding, a variation of ordinary finite-state coding. At each state u of the encoder, the allowable input tag sequences are not arbitrary; namely, there is a list of n-ary words $W(u)$, all of the same length K, such that an input tag sequence labeling a path from u is allowable if and only if its K-prefix belongs to $W(u)$. So, when we encode an input tag (an input symbol), we do so with the commitment to encode thereafter only a certain specified collection of input tag sequences. The encoded input tag and the next encoder state are dictated by the upcoming sequence of input tags, subject to the requirement that each edge in the encoder graph is associated with exactly one input tag.

In order to describe look-ahead encoding precisely, we will make use of the following notation: for sets U and V of (finite) words, we denote by UV the set of all concatenations uv such that $u \in U$ and $v \in V$. We will use the notation B_n for the n-ary set $\{0, 1, \ldots, n-1\}$.

A *look-ahead encoder* (or more precisely, a *K-tag look-ahead encoder with input alphabet B_n*) for a lossless graph $G = (V, E, L)$ is defined by an input tagging $L_I : E \to B_n$ of the edges of G and subsets $W(u)$ of $(B_n)^K$ for each state $u \in V$ such that the following two conditions hold:

(LA-1) for at least one state $u \in V$ we have $W(u) \neq \emptyset$;

(LA-2) for each $u \in V$ we have

$$W(u)B_n \subseteq \cup_{e \in E_u} L_I(e) W(\tau_G(e)) \ .$$

Now, we can transform a look-ahead encoder into a generalized tagged $(S(G), n)$-encoder (and therefore an ordinary tagged $(S(G), n)$-encoder) as follows. For each state $u \in V$, $\mathbf{s} \in W(u)$, and $b \in B_n$, use condition (LA-2) above to choose a particular outgoing edge $e = e(u, \mathbf{s}b)$ from u in G such that $\mathbf{s}b = a\mathbf{s}'$, $a = L_I(e)$, and $\mathbf{s}' \in W(\tau_G(e))$. Now, construct a new graph $\mathcal{E} = (V', E', L'_I/L'_O)$ with two labelings—input labeling L'_I and output labeling L'_O—as follows. The states of $\mathcal{E}$ are all pairs $[u, \mathbf{s}]$, where $u \in V$ and $\mathbf{s} \in W(u)$. We draw an edge $[u, \mathbf{s}] \stackrel{a/c}{\to} [u', \mathbf{s}']$ (with L'_I-labeling a and L'_O-labeling c) if and only if the following three conditions hold:

(a) $\mathbf{s}b = a\mathbf{s}'$, where a is the first B_n-symbol in $\mathbf{s}$ and b is the last B_n-symbol in $\mathbf{s}'$.

(b) $u' = \tau_G(e(u, \mathbf{s}b))$.

(c) $c = L(e(u, \mathbf{s}b))$.

It can be shown that $\mathcal{E}$ is a generalized tagged (S, n)-encoder (Problem 6.5). Moreover, if G has finite anticipation (resp., has finite memory), then $\mathcal{E}$ has finite anticipation (resp., is sliding-block decodable). In particular, if G has finite memory $\mathcal{M}$, then $\mathcal{E}$ is sliding-block decodable with decoding window length $\leq \mathcal{M}+1$: any word of length $\mathcal{M}+1$ in $S(G)$ uniquely determines an edge e of G and the decoded input tag in $\mathcal{E}$ is then $L'_I(e)$. So, if G has memory zero, then the decoding window length is 1 (note that this does not mean that the encoder is block decodable, i.e., $(0, 0)$-sliding-block decodable).

A very special case of the main result in [AM95] is that for any K-tag look-ahead encoder for a labeled graph G, there is an equivalent encoder (in the sense that the encoding function is the same modulo a shift) obtained from G by at most K rounds of out-splitting. Now, if G has memory $\mathcal{M}$, then the upper bound in Section 5.5.2 given for the smallest decoding window length of a sliding-block decoder is $\mathcal{M}+K+1$. However, as mentioned above, such an encoder is sliding-block decodable with window length at most $\mathcal{M}+1$. So, here is an instance where the bound on the decoding window length given by the state-splitting algorithm is much larger than the actual decoding window length.

Lempel and Cohn [LemCo82] proposed a generalization of the look-ahead encoding method. Their method differs from look-ahead encoding in several respects. First, at each state, the set of input words is allowed to have variable lengths (this is not really an essential difference, and this possibility was already considered by Franaszek [Fra80a]). Second, the outgoing edge dictated by an input tag is allowed to depend on the sequence of previous input tags (as well as following input tags). Third, an edge in the encoder graph may be associated with more than one input tag. While this still gives a well-defined encoder, it may not have finite anticipation, and so decoding can be problematic. Nevertheless, by a series of examples, Lempel and Cohn showed that this method works very well in many circumstances.

6.6 Bounded-delay encoders

The bounded-delay method is a generalization of look-ahead encoding. Let $G = (V, E, L)$ be a labeled graph. The *Tth order Moore form of G*, denoted $G^{\{T\}}$, is the labeled graph defined as follows. The states of $G^{\{T\}}$ are the paths of length $T-1$ in G, and for each path $e_1e_2 \ldots e_T$ of length T in G, there is an edge $e_1e_2 \ldots e_{T-1} \rightarrow e_2e_3 \ldots e_T$ in $G^{\{T\}}$ labeled $L(e_T)$. The labeled graph $G^{\{2\}}$ is the ordinary Moore form of G, as was defined in Section 2.2.7.

Let $G = (V, E, L)$ be a lossless presentation of a constrained system S, and let K and T be positive integers. A *bounded-delay encoder* (or more precisely, a *K-tag, delay-T bounded-delay encoder with input alphabet B_n*) for a lossless graph G is a K-tag look-ahead encoder with input alphabet B_n for the Tth order Moore form $G^{\{T\}}$ of G. Observe that if G has finite memory $\mathcal{M}$, then such an encoder is sliding-block decodable with decoding window length $\mathcal{M}+T+1$. We remark that Franaszek [Fra82] originally framed his IP approach (discussed in Section 6.1) in terms of bounded-delay encoding.

Hollmann [Holl95] has recently developed an approach which combines features of the bounded-delay method and the state-splitting method. The idea is to first split states sufficiently until one obtains a graph which is amenable to a certain variation of the bounded-delay method. In many cases, the result of this procedure is a sliding-block decodable encoder whose decoder has smaller window length than would be expected. Several very nice codes for specific constraints of practical importance were constructed in [Holl95].

Hollmann's approach was influenced by earlier work of Immink [Imm92], as well as by Franaszek's bounded-delay encoding method. Immink showed that in many situations, while there may be no block decodable tagged (S, n)-encoder, it may still be possible to construct a tagged (S, n)-encoder which is $(-1, 1)$-sliding-block decodable. The decoder produces a decoded input tag at time i, by examining only the output symbol at time $i+1$; in other words, it decodes by looking into the future and ignoring the present. What does this mean in terms of the input tagging of an (S, n)-encoder $\mathcal{E}$? Well, suppose, for simplicity, that $\mathcal{E}$ has memory 1 and at each state, the input tags on the *incoming* edges are all the same. Then, any output symbol uniquely determines a state v in $\mathcal{E}$ and is decoded to the input tag which is written on the incoming edges to v. Of course, it is not at all obvious how to construct such encoders. However, useful heuristics, illustrated by several examples, were given in [Imm92], as well as in subsequent papers by Immink [Imm95a] and Hollmann [Holl94]. In [Holl95], [Holl96], Hollmann develops codes that look 'further' into the future: in his construction, he aims for $(-\mathsf{m}, \mathsf{a})$-sliding-block decodable encoders, where $0 \leq \mathsf{m} \leq \mathsf{a}$; that is, the decoder produces a decoded input tag at time i by examining only the output symbols $w_{i+\mathsf{m}} w_{i+\mathsf{m}-1} \ldots w_{i+\mathsf{a}}$.

6.7 Transforming a generalized encoder to an ordinary encoder

Here, we outline how to transform a generalized tagged (S, n)-encoder to an ordinary tagged (S, n)-encoder.

Let G be a labeled graph with finite anticipation $\mathcal{A}$. We define the *induced labeled graph* $G' = (V', E', L')$ of G in the following manner. The states of G' are all pairs $[u, \mathbf{s}]$, where $u \in V_G$ and $\mathbf{s}$ is a word of length $\mathcal{A}$ that can be generated from u in G. We draw an edge $[u, \mathbf{s}] \xrightarrow{b} [u', \mathbf{s}']$ if and only if the following two conditions hold:

(a) $\mathbf{s}b = a\mathbf{s}'$, where a is the first symbol in $\mathbf{s}$;

(b) u' is the terminal state of the first (unique) edge $u \xrightarrow{a} u'$, denoted $e(u, \mathbf{s}b)$, in any path of length $\mathcal{A}+1$ that is labeled by the word $\mathbf{s}b = a\mathbf{s}'$ in G starting at state u.

Kitchens [Kit81] (see also [BKM85]) showed that whenever $G = (V, E, L)$ is a labeled graph with finite anticipation, then the corresponding induced labeled graph G' is deterministic and $S(G) = S(G')$ (Problem 6.6).

Now, given a generalized tagged (S, n)-encoder $\mathcal{E} = (V, E, L_I/L_O)$, we apply the preceding result to $\mathcal{E}_I = (V, E, L_I)$, yielding a deterministic induced tagged graph $\mathcal{E}'_I = (V', E', L'_I)$ (i.e., the labels given by L'_I are regarded as "input tags"). We form a generalized tagged (S, n)-encoder $\mathcal{E}' = (V', E', L'_I/L'_O)$ by endowing $\mathcal{E}'_I$ with an output labeling L'_O to reflect the labeling L_O: namely, the L'_O-label, c, of the edge $[u, \mathbf{s}] \xrightarrow{b/c} [u', \mathbf{s}']$ in $\mathcal{E}'$ is set to $c = L_O(e(u, \mathbf{s}b))$.

Clearly, both $\mathcal{E}_I$ and $\mathcal{E}'_I$ present the set of all n-ary words through their labelings L_I and L'_I. Hence, since $\mathcal{E}'_I$ is deterministic, there is an (input-)labeled subgraph $\mathcal{E}''_I = (V'', E'', L'_I)$ of $\mathcal{E}'_I$ with constant out-degree n which still presents the unconstrained system of n-ary words. Now, by the losslessness of $\mathcal{E}_O = (V, E, L_O)$ we have that the labeled graph $\mathcal{E}'_O = (V', E', L'_O)$ and, therefore, also $\mathcal{E}''_O = (V'', E'', L'_O)$, is an (S, n)-encoder. The tagging L'_I then converts it into an (ordinary) tagged (S, n)-encoder $\mathcal{E}'' = (V'', E'', L'_I/L'_O)$. In this way, we have transformed the generalized tagged (S, n)-encoder $\mathcal{E} = (V, E, L_I/L_O)$ to the ordinary tagged (S, n)-encoder $\mathcal{E}''$ (although the encoding mapping is shifted in the process). Moreover, if $\mathcal{E}_O$ has finite anticipation, then the resulting (S, n)-encoder $\mathcal{E}''_O$ will have finite anticipation, and if $\mathcal{E}_O$ has finite memory, then the resulting tagged (S, n)-encoder $\mathcal{E}''$ will be sliding-block decodable. Therefore, the notions of finite anticipation and sliding-block decodability can be applied to generalized encoders.

The induced labeled graph described above resembles that produced by the determinizing construction of Section 2.2.1. However, it is different: the determinizing construction yields a labeled graph which is 'too small' to allow the definition of a well-defined labeling L'_O that reflects the labeling L_O.

Problems

Problem 6.1 Show that the graph G', constructed in Section 6.1, is lossless.

Problem 6.2 Show that $(V, E, L_I/L)$, as described at the end of Section 6.4.1, is a generalized tagged (S, n)-encoder (when viewed as an ordinary graph with two labelings—input tagging L_I and output labeling L).

Problem 6.3 Let $G = (V, E, L)$ be a variable-length graph (VLG). Denote by $Q_G(z)$ the $|V| \times |V|$ matrix in the indeterminate z, with entries given by

$$(Q_G(z))_{u,v} = \sum_e z^{-\ell(e)} ,$$

where $u, v \in V$ and the summation is taken over all edges e in G that originate in u and terminate in v.

Define the *ordinary form of* G as the ordinary graph H obtained from G by replacing each edge e, of length $\ell(e)$, with a chain of $\ell(e)$ edges connected through $\ell(e)-1$ new states.

1. Show that if G is an ordinary graph (i.e., $\ell(e) = 1$ for every $e \in E$), then
$$Q_G(z) = z^{-1} A_G ,$$
where A_G is the adjacency matrix of G.
2. Show that if H is the ordinary form of a VLG G, then μ is a nonzero eigenvalue of A_H if and only if
$$\det(Q_G(\mu) - I) = 0 .$$
In particular, the largest real solution of $\det(Q_G(z) - I) = 0$ is $z = \lambda(A_H)$.

Problem 6.4 A finite set Γ of words over an alphabet Σ is called *exhaustive* if every word $\mathbf{w}$ over Σ is either a prefix of a word in Γ or there is a word in Γ that is a prefix of $\mathbf{w}$. Denote by $\ell(\mathbf{w})$ the length of the word $\mathbf{w}$.

1. Let Γ be an exhaustive set of words over an alphabet of size n. Show that
$$\sum_{\mathbf{w} \in \Gamma} n^{-\ell(\mathbf{w})} \geq 1 .$$
Hint: Let $\ell_{\max}$ be the largest length of any word in Γ; show that
$$\sum_{\mathbf{w} \in \Gamma} n^{\ell_{\max} - \ell(\mathbf{w})} \geq n^{\ell_{\max}} .$$

2. Let $m_1, m_2, \ldots, m_t$ be nonnegative integers that satisfy

$$\sum_{i=1}^{t} n^{-m_i} \geq 1 \ .$$

Show that there exist integers $\ell_1, \ell_2, \ldots, \ell_t$ such that $\ell_i \geq m_i$ and

$$\sum_{i=1}^{t} n^{-\ell_i} = 1 \ .$$

3. A finite set Γ of words over am alphabet Σ is called *prefix-free* is no word in Γ is a prefix of any other word in Γ. Show that if Γ is prefix-free, then

$$\sum_{\mathbf{w} \in \Gamma} n^{-\ell(\mathbf{w})} \leq 1 \ .$$

4. Let $\ell_1, \ell_2, \ldots, \ell_t$ be nonnegative integers that satisfy the equality

$$\sum_{i=1}^{t} n^{-\ell_i} = 1 \ .$$

Suggest an efficient algorithm for constructing a set Γ of t words over an alphabet of size n such that the following three conditions hold:

(a) Γ is exhaustive;

(b) Γ is prefix-free; and —

(c) the ith word in Γ has length ℓ_i.

Problem 6.5 Let $\mathcal{E}$ be the encoder, based on a lossless graph G, described in Section 6.5.

1. Show that $\mathcal{E}$ is a generalized tagged (S, n)-encoder.

2. Show that if G has finite anticipation (resp., has finite memory), then $\mathcal{E}$ has finite anticipation (resp., is sliding-block decodable).

Problem 6.6 Let G be a labeled graph with finite anticipation, and let G' be the induced labeled graph described in Section 6.7.

1. Show that G' is deterministic.

2. Show that $S(G) = S(G')$.