Chapter 11

Codes on graphs

In this chapter we will introduce the subject of codes on graphs. This subject forms an intellectual foundation for all known classes of capacity-approaching codes, including turbo codes and low-density parity-check (LDPC) codes.

There are many styles of graphical realizations of codes; *e.g.,* parity-check realizations (Tanner graphs), generator realizations, and trellis (state) realizations. More generally, we consider "behavioral" realizations, in which the time axis of a trellis realization is replaced by a general unordered graph.

We will first consider elementary linear behavioral realizations, in which the constraints are given by sets of linear equations. We then generalize to cases in which the constraints are given by linear codes. We show how behavioral realizations may be naturally represented by graphical models of various types: in particular, Tanner graphs and normal (Forney) graphs. More general classes of graphical models (*e.g.,* factor graphs, Maarkov graphs, block diagrams, and Bayesian networks) are discussed briefly in an Appendix.

Finally, we develop some elementary but important properties of graphical realizations, particularly the cut-set bound, which builds on the state space theorem.

11.1 Elementary realizations of linear block codes

We continue to restrict attention to linear (n, k) block codes over a finite field $\mathbb{F}_q$. So far we have seen several general methods of characterizing such codes:

- By a set of k generators $\{\mathbf{g}_j, 1 \leq j \leq k\}$. The code $\mathcal{C}$ is then the set of all linear combinations $\sum_i u_i \mathbf{g}_i$ of the generators over $\mathbb{F}_q$.
- By a set of $n-k$ generators $\{\mathbf{h}_j, 1 \leq j \leq n-k\}$ for the dual code $\mathcal{C}^\perp$. The code $\mathcal{C}$ is then the set of all n-tuples $\mathbf{y} \in (\mathbb{F}_q)^n$ such that $\langle \mathbf{y}, \mathbf{h}_j \rangle = 0$ for all j.
- By a trellis (state-space) realization. The code is then the set of all n-tuples corresponding to paths through the trellis.

We will see that these realizations are all special cases of a general class of realizations called *behavioral realizations* (from the behavioral approach to system theory pioneered by Willems). In general, a behavioral realization defines a code by a set of constraints that the code symbols and other auxiliary state variables must satisfy.

For linear codes, we need consider only linear behavioral realizations, where the variables are over a field and the constraints are linear. In the simplest case, the variables are field elements and the constraints are linear equations involving the variables. In the general case, the variables can be vector spaces over the field, and the constraints are expressed in terms of linear codes.

11.1.1 Elementary linear behavioral realizations

The elements of an elementary linear behavioral realization of a linear (n, k, d) block code over $\mathbb{F}_q$ are as follows:

- The n code symbols $\mathbf{y} = \{y_i \in \mathbb{F}_q, i \in \mathcal{I}\}$, where $\mathcal{I}$ denotes the symbol index set.
- An additional set of s auxiliary variables $\mathbf{s} = \{s_j \in \mathbb{F}_q, j \in \mathcal{J}\}$, often called state (hidden, latent, unobserved) variables, where the state variable index set $\mathcal{J}$ may be unrelated to $\mathcal{I}$.
- A set of e linear homogeneous equations over $\mathbb{F}_q$ involving the components of the symbol and state variables, called the constraint equations.

The *full behavior* $\mathfrak{B}$ generated by the realization is the set of all combinations $(\mathbf{y}, \mathbf{s})$ (called *trajectories*) of symbol and state variables that satisfy all constraint equations. The code $\mathcal{C}$ generated by the realization is the set of all symbol n-tuples $\mathbf{y}$ that appear in any trajectory $(\mathbf{y}, \mathbf{s}) \in \mathfrak{B}$; *i.e.,* such that there exists some set $\mathbf{s}$ of state variables such that $(\mathbf{y}, \mathbf{s}) \in \mathfrak{B}$.

In general, the e linear homogeneous constraint equations may be written in matrix form as

$$\mathbf{y}A + \mathbf{s}B = \mathbf{0},$$

where $\mathbf{y}$ is a row n-tuple of symbols, $\mathbf{s}$ is a row s-tuple of state variable components, and A and B are $n \times e$ and $s \times e$ $\mathbb{F}_q$-matrices, respectively. The set $\mathfrak{B}$ of all solutions $(\mathbf{y}, \mathbf{s})$ to such a set of equations is a subspace of the vector space $(\mathbb{F}_q)^{n+s}$ of dimension $\dim \mathfrak{B} \geq n + s - e$, with equality if and only if all equations are linearly independent.

The code $\mathcal{C}$ is the projection of $\mathfrak{B}$ onto its first n components. The dimension of $\mathcal{C}$ is equal to the dimension of $\mathfrak{B}$ if and only if codewords corresponding to distinct trajectories are distinct.

We now show that generator matrices and parity-check matrices yield elementary behavioral realizations of this kind.

Example 1 (generator realizations). Let G be a $k \times n$ generator matrix for $\mathcal{C}$, whose k rows form a set of linearly independent generators for $\mathcal{C}$. Then $\mathcal{C}$ is the set of all n-tuples of the form $\mathbf{y} = \mathbf{u}G$ for some information k-tuple $\mathbf{u} \in (\mathbb{F}_q)^k$. Thus $\mathcal{C}$ has an elementary linear behavioral realization with a state k-tuple $\mathbf{u}$ and n constraint equations, namely

$$\mathbf{y} - \mathbf{u}G = \mathbf{0}.$$

For example, in the previous chapter we found the following trellis-oriented generator matrix for the $(8, 4, 4)$ RM code:

$$\begin{bmatrix} 11110000 \\ 01011010 \\ 00111100 \\ 00001111 \end{bmatrix}. \tag{11.1}$$

This yields a linear behavioral realization with 4 state variables and 8 constraint equations, namely the following linear homogeneous equations over $\mathbb{F}_2$:

$$\begin{aligned} y_0 &= u_1; \\ y_1 &= u_1 + u_2; \\ y_2 &= u_1 + u_3; \\ y_3 &= u_1 + u_2 + u_3; \\ y_4 &= u_2 + u_3 + u_4; \\ y_5 &= u_3 + u_4; \\ y_6 &= u_2 + u_4; \\ y_7 &= u_4. \end{aligned} \tag{11.2}$$

□

Example 2 (parity-check realizations). Let H be an $(n-k) \times n$ generator matrix for $\mathcal{C}^\perp$. Then $\mathcal{C}$ is the set of all n-tuples that satisfy the $n-k$ constraint equations

$$\mathbf{y}H^T = \mathbf{0}.$$

This corresponds to an elementary linear behavioral realization with no state variables.

For example, since the $(8, 4, 4)$ code $\mathcal{C}$ is self-dual, the generator matrix (11.1) is also a generator matrix for $\mathcal{C}^\perp$. This yields an elementary linear behavioral realization with no state variables and 4 constraint equations, namely the following linear homogeneous equations over $\mathbb{F}_2$:

$$\begin{aligned} y_0 + y_1 + y_2 + y_3 &= 0; \\ y_1 + y_3 + y_4 + y_6 &= 0; \\ y_2 + y_3 + y_4 + y_5 &= 0; \\ y_4 + y_5 + y_6 + y_7 &= 0. \end{aligned} \tag{11.3}$$

This is evidently a more compact realization than the generator realization of Example 1— in fact, it can be found by eliminating the state variables from Eqs. (11.2)— and, because it has no state variables, it is better suited for checking whether a given 8-tuple $\mathbf{y}$ is in $\mathcal{C}$. On the other hand, in the generator realization the state 4-tuple $\mathbf{u}$ may be freely chosen and determines the codeword— *i.e.,* the generator realization is an input-output realization— so it is better for generating codewords, *e.g.,* in encoding or in simulation. □

11.1.2 Graphs of elementary linear behavioral realizations

We may draw a graph of an elementary linear behavioral realization as follows. In coding theory, such a graph is called a *Tanner graph*.

The graph has two types of vertices, namely $n + s$ vertices corresponding to the n symbol and s state variables, and e vertices corresponding to the e constraint equations. An edge is drawn between a variable vertex and a constraint vertex if the corresponding variable is involved in the corresponding constraint. Thus the graph is *bipartite*; *i.e.,* the vertices are partitioned into two sets such that every edge connects a vertex of one type to one of the other type.

A generic Tanner graph therefore has the form of Figure 1(a). Here symbol variables are represented by filled circles, state variables by open circles, and constraints by squares containing a "+" sign, since all constraint equations are zero-sum (parity-check) constraints.

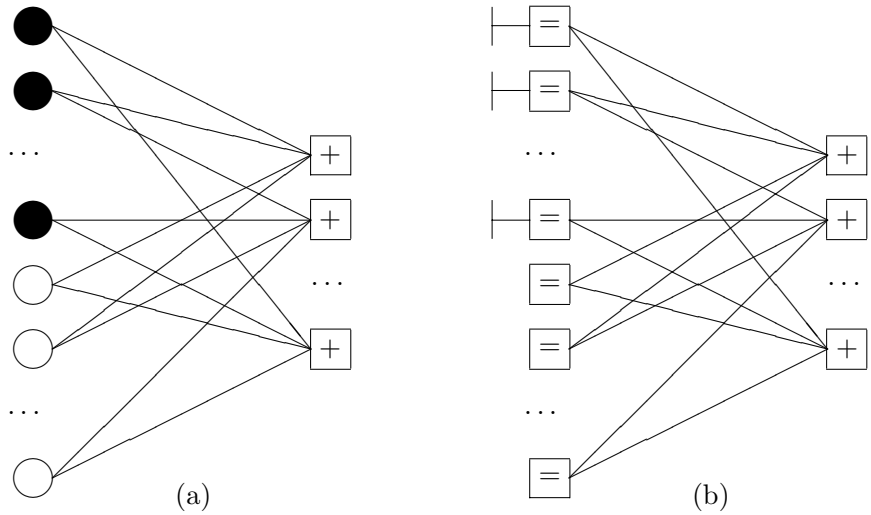

Figure 1. (a) Generic bipartite Tanner graph, with symbol variables (filled circles), state variables (open circles), and zero-sum constraints (squares with "+"). (b) Equivalent normal graph, with equality constraints replacing variables, and observed variables indicated by "dongles."

Figure 1(b) shows an equivalent normal graph (also called a Forney graph). Here the variables are replaced by equality constraints, so that all graph vertices represent constraints. Variables are represented by edges; an equality constraint ensures that all of its incident edges represent a common variable (as the edges in Tanner graphs do implicitly). Finally, a symbol variable is indicated by a special "half-edge" ("dongle") symbol incident on the corresponding equality constraint. The dongle may be regarded as an input/output terminal that can connect the corresponding symbol variable with the outside world, whereas state variables are hidden, internal variables that do not interact with the outside world.

The *degree* of a variable or equation will be defined as the degree (number of incident edges) of the corresponding graph vertex— *i.e.,* the degree of a variable is the number of equations that it is involved in, and the degree of an equation is the number of variables that it involves. In a Tanner graph, the sum of the variable degrees is equal to the sum of the constraint degrees, since both are equal to the number of edges in the graph. In a normal graph, if we choose not to count half-edges in vertex degrees, then the vertex degrees are the same.

Example 1 (generator realizations) (cont.) The Tanner graph corresponding to the generator realization of the $(8, 4, 4)$ code defined by Eqs. (11.2) is shown in Figure 2(a). Since each symbol variable has degree 1 in this realization, the corresponding symbol vertex is located adjacent to the unique constraint vertex with which it is associated. The equivalent normal graph is shown in Figure 2(b); here symbol variables may simply be represented by dongles.

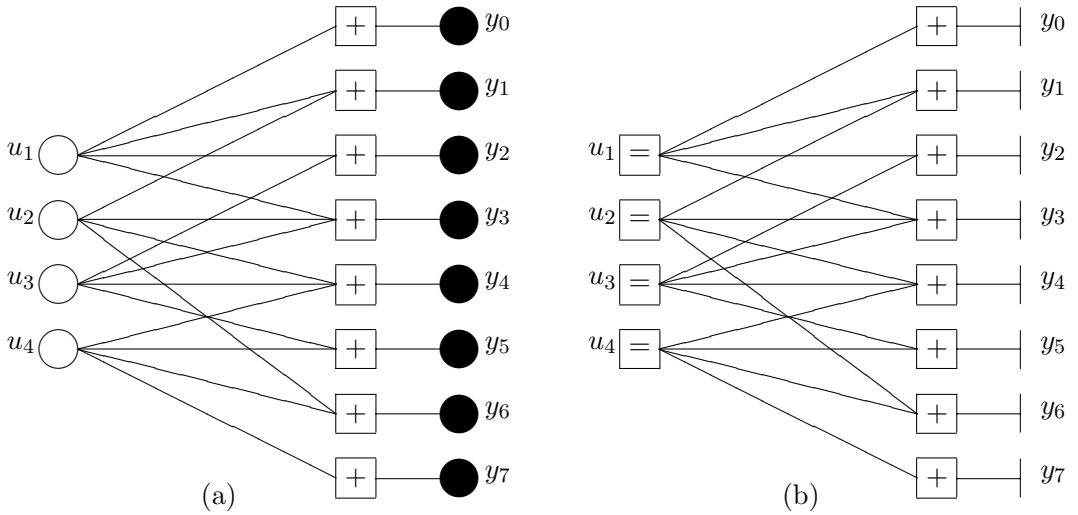

Figure 2. Generator realizations for $(8, 4, 4)$ code. (a) Tanner graph. (b) Normal graph.

Example 2 (parity-check realizations) (cont.) The Tanner graph and normal graph of the parity-check realization of the $(8, 4, 4)$ code defined by Eqs. (11.3) are shown in Figure 3.

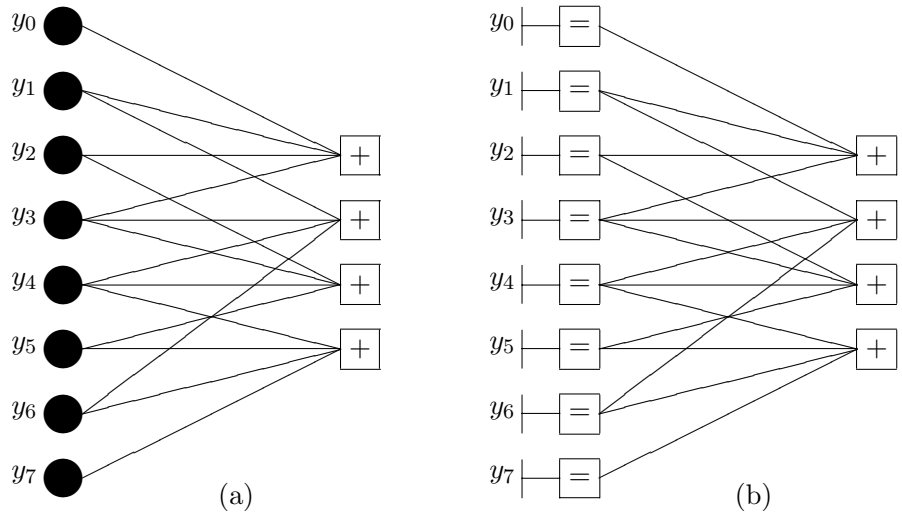

Figure 3. Parity-check realizations for $(8, 4, 4)$ code. (a) Tanner graph. (b) Normal graph.

The normal graphs of Figures 2(b) and 3(b) are duals, in the sense that one is obtained from the other by replacing equality constraints by zero-sum constraints and *vice versa*. In general, the dual of a generator realization for $\mathcal{C}$ will be a parity-check realization for $\mathcal{C}^\perp$, and *vice versa*; here we have $\mathcal{C} = \mathcal{C}^\perp$, since the $(8, 4, 4)$ code is self-dual. This illustrates an important general duality property of normal graphs, which we will not prove here: the dual of a normal graph realization of a code is a realization of the dual code. □

11.2 General linear behavioral realizations

We now generalize the elementary realizations above by letting symbol and state variables be vector spaces of dimension m over $\mathbb{F}_q$, or more particularly m-tuples over $\mathbb{F}_q$, where the dimension m may be different for each variable. Furthermore, we generalize to constraints that certain subsets of variables must lie in certain small linear block codes over $\mathbb{F}_q$.

The elements of a general linear behavioral realization of a linear (n, k) block code over $\mathbb{F}_q$ are therefore as follows:

- A set of symbol m_i-tuples $\{y_i \in (\mathbb{F}_q)^{m_i}, i \in \mathcal{I}\}$, where $\mathcal{I}$ denotes the symbol variable index set. We define $n = \sum_{\mathcal{I}} m_i$.
- A state index set $\mathcal{J}$, and a set of state spaces $\Sigma_j, j \in \mathcal{J}$, where Σ_j is a vector space over $\mathbb{F}_q$ of dimension μ_j. Such a state space Σ_j may always be represented by a vector space of μ_j-tuples, $\{\Sigma_j = (\mathbb{F}_q)^{\mu_j}, j \in \mathcal{J}\}$. We define $s = \sum_{\mathcal{J}} \mu_j$.
- A set of linear constraint codes $\{\mathcal{C}_k, k \in \mathcal{K}\}$ over $\mathbb{F}_q$, where each code $\mathcal{C}_k$ involves a certain subset of the symbol and state variables, and $\mathcal{K}$ denotes the constraint index set.

Again, the *full behavior* $\mathfrak{B}$ generated by the realization is the set of all trajectories $(\mathbf{y}, \mathbf{s})$ such that all constraints are satisfied— *i.e.,* such that for each k the values taken by the subset of variables involved in the constraint code $\mathcal{C}_k$ forms a codeword in $\mathcal{C}_k$— and the code $\mathcal{C}$ generated by the realization is the set of all symbol sequences $\mathbf{y}$ that appear in any trajectory $(\mathbf{y}, \mathbf{s}) \in \mathfrak{B}$.

Notice that the constraint imposed by a zero-sum constraint constrains the d variables incident on a zero-sum vertex of degree d to lie in the $(d, d-1, 2)$ zero-sum (SPC) code over $\mathbb{F}_q$. Similarly, an equality constraint of degree d constrains the d incident variables to lie in the $(d, 1, d)$ repetition code over $\mathbb{F}_q$. Thus allowing each constraint code $\mathcal{C}_k$ to be an arbitrary linear code generalizes the elementary linear behavioral realizations discussed earlier.

The generalization to variables of dimension m allows us to consider state spaces of dimension larger than 1, which we need for general trellis (state-space) realizations. It also allows us to consider the clustered symbol variables that arise in sectionalized trellis realizations.

We now show how trellis (state-space) realizations may be expressed as general linear behavioral realizations.

Example 3 (trellis realizations). Let us consider an unsectionalized minimal trellis realization of an (n, k) binary linear block code $\mathcal{C}$ on the time axis $\mathcal{I} = [0, n)$.

As we saw in the previous chapter, a minimal trellis realization of $\mathcal{C}$ may be defined by a trellis-oriented generator matrix G for $\mathcal{C}$ comprising k minimal-span generators $\{\mathbf{g}_j, 1 \le j \le k\}$. We thus have a one-to-one correspondence $\mathcal{C} \leftrightarrow (\mathbb{F}_2)^k$ defined by $\mathbf{u}G \leftrightarrow \mathbf{u}$.

We need to define an index set $\mathcal{J}$ for the state spaces $\Sigma_j, j \in \mathcal{J}$. When the symbol time index set is $\mathcal{I} = [0, n)$, we define the state index set as $\mathcal{J} = [0, n]$, with the understanding that the kth symbol comes after the kth state and before the $(k+1)$st state. The initial and final state spaces Σ_0 and Σ_n have dimension 0; *i.e.,* they are trivial.

We further need to define an explicit realization for the state spaces Σ_k. Let $\mathcal{J}(k)$ denote the set of indices of the trellis-oriented generators $\mathbf{g}_j$ that are active at state time k. The state code $\mathcal{S}_k$ at state time k is then generated by the submatrix $G_k = \{\mathbf{g}_j, j \in \mathcal{J}(k)\}$. We thus have a one-to-one correspondence $\mathcal{S}_k \leftrightarrow (\mathbb{F}_2)^{|\mathcal{J}(k)|}$ defined by $\mathbf{u}_{|\mathcal{J}(k)} G_k \leftrightarrow \mathbf{u}_{|\mathcal{J}(k)}$, where $\mathbf{u}_{|\mathcal{J}(k)} = \{u_j, j \in \mathcal{J}(k)\}$.

Thus if we define a state space Σ_k whose alphabet is the set $(\mathbb{F}_2)^{|\mathcal{J}(k)|}$ of $|\mathcal{J}(k)|$-tuples $\mathbf{u}_{|\mathcal{J}(k)}$, then we obtain a state space Σ_k of minimal dimension $|\mathcal{J}(k)|$ such that any codeword associated with a state codeword $\mathbf{u}_{|\mathcal{J}(k)}G_k \in \mathcal{S}_k$ passes through the state $\mathbf{u}_{|\mathcal{J}(k)} \in \Sigma_k$, as desired.

The branch space $\mathcal{B}_k$ at symbol time $k \in [0, n)$ is then the set of all $(|\mathcal{J}(k)|+|\mathcal{J}(k+1)|+1)$-tuples $(\sigma_k, y_k, \sigma_{k+1})$ that can actually occur. If $\mathcal{K}(k)$ denotes the subset of trellis-oriented generators that are active at symbol time k, then by a similar development it can be seen that $\mathcal{B}_k$ is a one-to-one linear function of $\mathbf{u}_{|\mathcal{K}(k)}$, so $\dim \mathcal{B}_k = |\mathcal{K}(k)|$. We may thus view $\mathcal{B}_k$ as a linear constraint code of length $|\mathcal{J}(k)| + |\mathcal{J}(k+1)| + 1$ and dimension $|\mathcal{K}(k)|$.

In summary, the elements of an unsectionalized minimal trellis realization of an (n, k) binary linear block code $\mathcal{C}$ are therefore as follows:

- A set of binary symbol variables $\{Y_k, k \in [0, n)\}$;
- A set of state spaces $\{\Sigma_k, k \in [0, n]\}$ of dimension $|\mathcal{J}(k)|$, represented by binary $|\mathcal{J}(k)|$-tuples, where $\{\mathbf{g}_j, j \in \mathcal{J}(k)\}$ is the subset of trellis-oriented generators that are active at state time $k \in [0, n]$;
- A set of binary linear constraint codes $\{\mathcal{B}_k, k \in [0, n)\}$, where $\mathcal{B}_k \subseteq \Sigma_k \times Y_k \times \Sigma_{k+1}$ and $\dim \mathcal{B}_k = |\mathcal{K}(k)|$, where $\{\mathbf{g}_j, j \in \mathcal{J}(k)\}$ is the subset of trellis-oriented generators that are active at symbol time $k \in [0, n)$.

The full behavior $\mathfrak{B}$ of the trellis realization is then the set of all state/symbol sequences $(\mathbf{s}, \mathbf{y})$ such that $(s_k, y_k, s_{k+1}) \in \mathcal{B}_k$ for $k \in [0, n)$. For each state/symbol sequence $(\mathbf{s}, \mathbf{y})$ in $\mathfrak{B}$, the state sequence $\mathbf{s}$ represents a valid path through the code trellis, and the symbol sequence $\mathbf{y}$ represents the corresponding codeword. If the trellis is minimal, then each path $(\mathbf{s}, \mathbf{y})$ corresponds to a distinct codeword $\mathbf{y}$, so $|\mathfrak{B}| = |\mathcal{C}|$.

Continuing with our $(8, 4, 4)$ example, its trellis-oriented generators (11.1) are active during $[0, 3), [1, 6), [2, 5)$ and $[4, 7)$, respectively. Therefore the state space dimension profile is $|\mathcal{J}(k)| = \{0, 1, 2, 3, 2, 3, 2, 1, 0\}$, and the branch space dimension profile is $|\mathcal{K}(k)| = \{1, 2, 3, 3, 3, 3, 2, 1\}$.

Figure 4(a) shows a Tanner graph of this minimal trellis realization, and Figure 4(b) is an equivalent normal graph. Each state space Σ_k is labelled by its dimension. The state spaces Σ_0 and Σ_8 do not need to be shown, because they are trivial and not actually involved in any constraints. Each constraint code (branch space) $\mathcal{B}_k$ is labelled by its length and dimension. Since the symbol variables Y_k have degree 1, we use the special "dongle" symbol for them. Since the state spaces Σ_k have degree 2, they are naturally represented by edges in a normal graph.

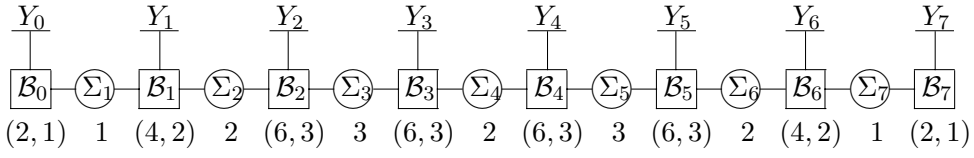

Figure 4(a). Tanner graph for minimal trellis realization of $(8, 4, 4)$ code.

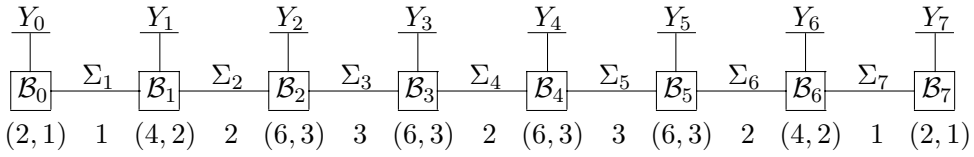

Figure 4(b). Equivalent normal graph for minimal trellis realization of $(8, 4, 4)$ code.

Every trellis (state-space) realization has a similar chain graph, with constraint codes $\mathcal{B}_k$ constraining one symbol variable Y_k and two state variables, Σ_k and Σ_{k+1}. More generally, the symbol variables Y_k may have arbitrary dimension; *i.e.,* symbols may be clustered. Note that in any trellis realization all symbol variables have degree 1, and all nontrivial state variables have degree 2. Note also that a trellis graph has no cycles (loops). □

11.3 Graph-theoretic properties of graphical realizations

A graph illustrates dependency relationships. We now develop some elementary but important connections between graph properties and dependencies.

11.3.1 Connectedness and independence

Suppose that a code $\mathcal{C}$ is the Cartesian product of two codes, $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$. In other words, $\mathcal{C}$ consists of the pairs of codewords $(\mathbf{c}_1, \mathbf{c}_2)$ such that $\mathbf{c}_1 \in \mathcal{C}_1, \mathbf{c}_2 \in \mathcal{C}_2$. Then a realization of $\mathcal{C}$ may be constructed from independent realizations of $\mathcal{C}_1$ and $\mathcal{C}_2$. A graph of such a realization is a disconnected graph, with two component subgraphs representing $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively.

Conversely, if a graph of a realization of $\mathcal{C}$ is disconnected, then $\mathcal{C}$ is evidently the Cartesian product of the codes realized by the component subgraphs. In short, a code $\mathcal{C}$ has a realization whose graph is disconnected if and only if $\mathcal{C}$ is the Cartesian product of shorter codes. Thus disconnectedness is a graph-theoretic expression of independence.

11.3.2 Cut sets and conditional independence

A *cut set* of a connected graph is a minimal set of edges such that removal of the set partitions the graph into two disconnected subgraphs.

Notice that a connected graph is *cycle-free* if and only if every edge is by itself a cut set.

In a normal graph, a cut set consists of a set of ordinary (state) edges, and may be specified by the corresponding subset $\chi \subseteq \mathcal{J}$ of the state index set $\mathcal{J}$. If the cut set consists of a single edge, then the cut set may be identified with the corresponding state variable Σ_j. If the cut set consists of several edges, then it may be identified with the set of all corresponding state spaces $\Sigma_j, j \in \chi$. We will regard the Cartesian product of all these state spaces as a superstate variable $\Sigma_\chi = \prod_{j\in\chi} \Sigma_j$. Note that the size of the alphabet of Σ_χ is $|\Sigma_\chi| = \prod_{j\in\chi} |\Sigma_j|$, the product of the sizes of its component state spaces.

Figure 5 gives a high-level view of a realization with a cut set χ. Since removal of a cut set partitions a graph into two disconnected subgraphs, it follows that the symbol variables, the constraint codes, and the states not in χ are partitioned by the cut set into two disjoint subsets connected only by the states in χ. We label these two components arbitrarily as the "past" $\mathcal{P}$ and the "future" $\mathcal{F}$ relative to the cut set χ. The two subsets of symbol variables associated with the past and future components are denoted by $Y_{|\mathcal{P}}$ and $Y_{|\mathcal{F}}$, respectively. The states in the cut set are regarded as a single superstate variable $\Sigma_\chi = \{\Sigma_j, j \in \chi\}$, with values $\mathbf{s}_\chi = \{s_j, j \in \chi\}$. The constraints and internal variables in the past and future components are agglomerated into aggregate constraints $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{F}$ that jointly constrain the aggregate superstate variable Σ_χ and the aggregate symbol variables $Y_{|\mathcal{P}}$ and $Y_{|\mathcal{F}}$, respectively.

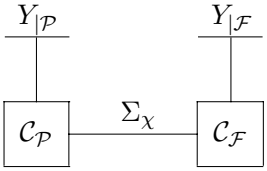

Figure 5. High-level view of a realization with a cut set χ.

Figure 5 makes it clear that a cut set in the graph corresponds to a certain conditional independence (Markov) property: given that the state variables in the cut set have a certain set of values $\mathbf{s}_\chi = \{s_j, j \in \chi\}$, the possible values of $Y_{|\mathcal{F}}$ depend only on $\mathbf{s}_\chi$ and not otherwise on the "past" $\mathcal{P}$, and *vice versa*. In other words, the superstate variable value $\mathbf{s}_\chi$ is a sufficient statistic for the past with respect to the set of possible futures, and *vice versa*.

More concretely, let $Y_{|\mathcal{P}}(\mathbf{s}_\chi)$ and $Y_{|\mathcal{F}}(\mathbf{s}_\chi)$ denote the sets of possible past and future symbol values that are consistent with a given superstate value $\mathbf{s}_\chi$, in view of the constraints $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{F}$. Then the set of possible codewords consistent with $\mathbf{s}_\chi$ is the Cartesian product $Y_{|\mathcal{P}}(\mathbf{s}_\chi) \times Y_{|\mathcal{F}}(\mathbf{s}_\chi)$. (In effect, fixing the value of the superstate removes the corresponding edge and disconnects the graph.) The set $\mathcal{C}$ of all possible codewords is then the union of such Cartesian products over all superstates:

$$\mathcal{C} = \bigcup_{\mathbf{s}_\chi \in \Sigma_\chi} Y_{|\mathcal{P}}(\mathbf{s}_\chi) \times Y_{|\mathcal{F}}(\mathbf{s}_\chi). \tag{11.4}$$

11.3.3 The cut-set bound

Observe that (11.4) is a generic expression for a code $\mathcal{C}$ generated by a two-section trellis with central state space Σ_χ, and that Figure 5 is a generic normal graph for such a two-section trellis. This observation leads directly to a lower bound on the size of Σ_χ:

Theorem 11.1 (Cut-set bound) *Given a graphical realization of a code $\mathcal{C}$ and a cut set χ, the size $|\Sigma_\chi| = \prod_{j\in\chi} |\Sigma_j|$ of the alphabet of the superstate variable $\Sigma_\chi = \{\Sigma_j, j \in \chi\}$ is lowerbounded by the minimal state space size in a conventional trellis in which the symbol variables are divided into "past" and "future" in the same way.*

If $\mathcal{C}$ is linear, then minimal state space size is given by the state space theorem for linear codes.

For example, by the cut-set bound and the Muder bound, given any graphical realization of the $(8, 4, 4)$ binary code and any cut set that partitions the code symbols into two subsets of size 4, the size of the alphabet of the superstate variable Σ_χ must be at least 4.

We can draw some important general conclusions from the cut-set bound.

First, consider cycle-free graphical realizations. In a cycle-free realization, every edge (state variable) is a cut set, and therefore the size of every state space is lowerbounded by the minimal size of a state space in some conventional trellis in which the symbol variables are partitioned into "past" and "future" in the same way. Therefore we cannot expect any great reduction in state space sizes from using general cycle-free graphs rather than conventional trellis realizations.

On the other hand, significant reductions in state space size are possible if we use graphs with cycles. Then cut sets will generally correspond to multiple state variables, and the complexity mandated by the cut-set lower bound may be spread out across these multiple state spaces.

We now illustrate these general conclusions by considering two particular styles of realizations: tail-biting trellis realizations, and Hadamard-transform-based realizations of Reed-Muller codes.

11.3.4 Tail-biting trellis realizations

A tail-biting trellis, illustrated in Figure 6, is a trellis in which the topology of the time axis is that of a circle rather than an interval. In other words, the ending state variable is also the starting state variable, and its alphabet may be of any size.

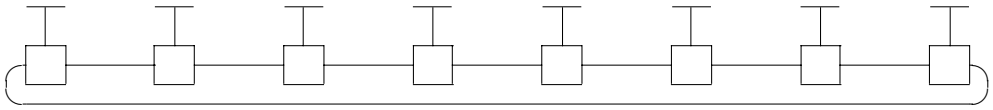

Figure 6. Normal graph of a tail-biting trellis realization.

In a tail-biting trellis realizations, all cut sets involve two state variables. Therefore the minimum complexity mandated by the cut-set bound may be spread out over two state spaces, each of which may be as small as the square root of the cut-set lower bound.

For a simple but not very impressive example, consider the $(8, 4, 4)$ code. Figure 7(a) shows a two-section, four-state conventional trellis realization which clusters four symbol bits at a time, which as we saw in Chapter 10 is an optimal sectionalization. Figure 7(b) shows a two-section tail-biting trellis realization with two 2-state state spaces, which in this case may be obtained merely by splitting the central state space of Figure 7(a). Thus the square root lower bound is achieved. Note however that while Figure 7(b) has smaller state spaces, it is no longer cycle-free.

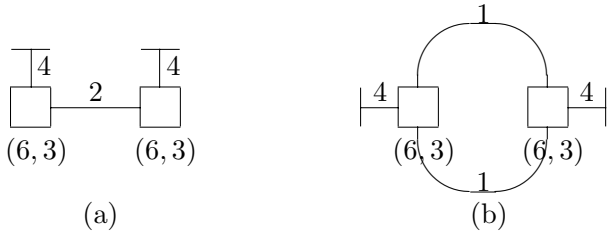

Figure 7. (a) Two-section, four-state trellis for $(8, 4, 4)$ code;
(b) Two-section, two-state tail-biting trellis realization.

More impressively, it has been shown that the state complexity of a 12-section realization of the $(24, 12, 8)$ Golay code may similarly be reduced from 256 (the minimum permitted by the Muder bound; see Exercise 10.6) to 16 by using a tail-biting realization. A "tail-biting trellis-oriented" generator matrix that yields this 16-state tail-biting realization is as follows:

11	01	11	01	11	00	00	00	00	00	00	00
00	11	11	10	01	11	00	00	00	00	00	00
00	00	11	01	10	11	11	00	00	00	00	00
00	00	00	11	01	11	01	11	00	00	00	00
00	00	00	00	11	01	11	01	11	00	00	00
00	00	00	00	00	11	11	10	01	11	00	00
00	00	00	00	00	00	11	01	10	11	11	00
00	00	00	00	00	00	00	11	01	11	01	11
11	00	00	00	00	00	00	00	11	01	11	01
01	11	00	00	00	00	00	00	00	11	11	10
10	11	11	00	00	00	00	00	00	00	11	01
01	11	01	11	00	00	00	00	00	00	00	11

Note that there are 4 "active" generators at each of the 12 state times, if we take the time axis to be circular ("end-around"). On the other hand, if we were to assume a conventional time axis, then at least 8 generators would have to be active at the central state time.

Note also that if we "unwrap" these generators onto an infinite conventional time axis, then we get generators for a rate-1/2 16-state period-4 time-varying (or rate-4/8 16-state time-invariant) binary linear convolutional code, as follows:

$$
\begin{array}{cccccccccccc}
 & & & & & \ldots & & & & & & \\
\ldots & 00 & 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 & \ldots \\
\ldots & 00 & 00 & 11 & 11 & 10 & 01 & 11 & 00 & 00 & 00 & \ldots \\
\ldots & 00 & 00 & 00 & 11 & 01 & 10 & 11 & 11 & 00 & 00 & \ldots \\
\ldots & 00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 & 00 & \ldots \\
 & & & & & \ldots & & & & & &
\end{array}
$$

This "Golay convolutional code" has a minimum Hamming distance of 8 and an average of $K_b = 12.25$ weight-8 codewords per information bit, so its nominal coding gain is $\gamma_c = 4$ (6 dB) and its effective coding gain is $\gamma_{\text{eff}} = 5.3$ dB, which are remarkable for a 16-state rate-1/2 code.

In summary, by considering a state realization with a single cycle rather than a conventional trellis realization, we may be able to obtain a state complexity as small as the square root of the minimum state complexity of a conventional trellis.

11.3.5 Hadamard-transform-based realizations of RM codes

In Exercise 6 of Chapter 6, it was shown that all Reed-Muller codes $\text{RM}(r, m)$ of length 2^m could be generated by a single "universal" $2^m \times 2^m$ generator matrix $U_m = (U_1)^{\otimes m}$, the m-fold tensor product of the 2×2 matrix $U_1 = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix} \right]$ with itself. The matrix U_m is called the *Hadamard transform* matrix over $\mathbb{F}_2$. For any binary 2^m-tuple $\mathbf{u} \in (\mathbb{F}_2)^{2^m}$, the binary 2^m-tuple $\mathbf{y} = \mathbf{u}U_m$ is called the *Hadamard transform* of $\mathbf{u}$. Since $(U_m)^2 = I_{2^m}$, the identity matrix, it follows that the Hadamard transform of $\mathbf{y}$ is $\mathbf{u}$; *i.e.,* $\mathbf{u} = \mathbf{y}U_m$

More particularly, $\text{RM}(r, m) = \{\mathbf{y} = \mathbf{u}U_m\}$, where the coordinates of the binary 2^m-tuple $\mathbf{u}$ are free in the $k(r, m)$ positions corresponding to the $k(r, m)$ rows of U_m of weight 2^{m-r} or greater, and fixed to 0 in the remaining coordinates. In other words, $\text{RM}(r, m)$ is the set of Hadamard transforms of all $2^{k(r,m)}$ binary 2^m-tuples that are all-zero in a certain $2^m - k(r, m)$ coordinates. (Compare the Fourier transform characterization of Reed-Solomon codes in Chapter 8.)

We can construct a graphical realization of a Hadamard transform as follows. The 2×2 Hadamard transform $\mathbf{y} = \mathbf{u}U_1$ is explicitly given by the two equations

$$
\begin{aligned}
y_0 &= u_0 + u_1; \\
y_1 &= u_1,
\end{aligned}
$$

which are realized by the normal graph of Figure 8. (This is sometimes called a controlled-NOT gate, where $y_1 = u_1$ is regarded as a control variable.)

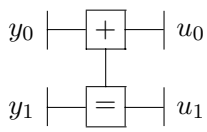

Figure 8. Normal graph of a 2×2 Hadamard transform.

Note that there are no arrows (directed edges) in this behavioral realization. Either $\mathbf{u}$ or $\mathbf{y}$ may be taken as input, and correspondingly $\mathbf{y}$ or $\mathbf{u}$ as output; *i.e.,* the graph is a realization of either the Hadamard transform $\mathbf{y} = \mathbf{u}U_1$ or the inverse Hadamard transform $\mathbf{u} = \mathbf{y}U_1$.

A $2^m \times 2^m$ Hadamard transform $\mathbf{y} = \mathbf{u}U_m$ may then be realized by connecting these 2×2 transforms in tensor product fashion. For example, the 8×8 Hadamard transform is given explicitly by the eight equations

$$\begin{aligned}
y_0 &= u_0 + u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_7; \\
y_1 &= u_1 + u_3 + u_5 + u_7; \\
y_2 &= u_2 + u_3 + u_6 + u_7; \\
y_3 &= u_3 + u_7; \\
y_4 &= u_4 + u_5 + u_6 + u_7; \\
y_5 &= u_5 + u_7; \\
y_6 &= u_6 + u_7; \\
y_7 &= u_7.
\end{aligned}$$

These equations are realized by the tensor product graph of Figure 9. (Compare the "butterflies" in the graph of an 8×8 fast Fourier transform.)

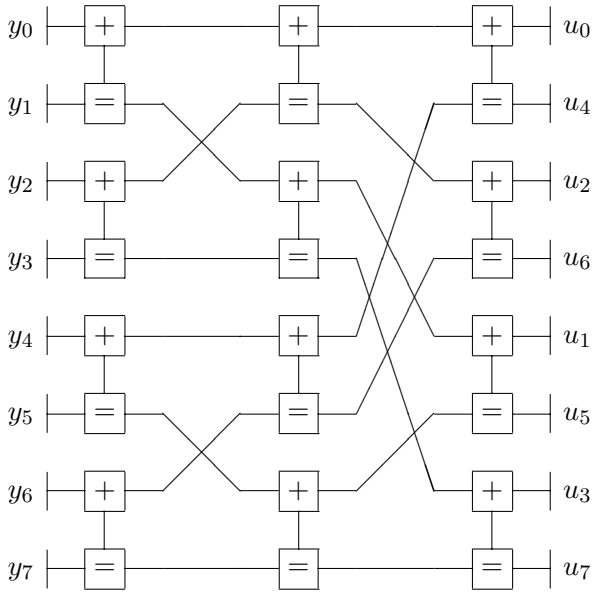

Figure 9. Normal graph of an 8×8 Hadamard transform.

A Reed-Muller code of length 8 may then be realized by fixing certain of the u_k to zero while letting the others range freely. For example, the $(8, 4, 4)$ code is obtained by fixing $u_0 = u_1 = u_2 = u_4 = 0$, which yields the equations

$$\begin{aligned}
y_0 &= u_3 + u_5 + u_6 + u_7; \\
y_1 &= u_3 + u_5 + u_7; \\
y_2 &= u_3 + u_6 + u_7; \\
y_3 &= u_3 + u_7;
\end{aligned}$$

$$\begin{aligned} y_4 &= u_5 + u_6 + u_7; \\ y_5 &= u_5 + u_7; \\ y_6 &= u_6 + u_7; \\ y_7 &= u_7. \end{aligned}$$

These equations are realized by the graph of Figure 10(a), which may be simplified to that of Figure 10(b). Here we regard the "inputs" u_j as internal variables, and the "outputs" y_k as external variables.

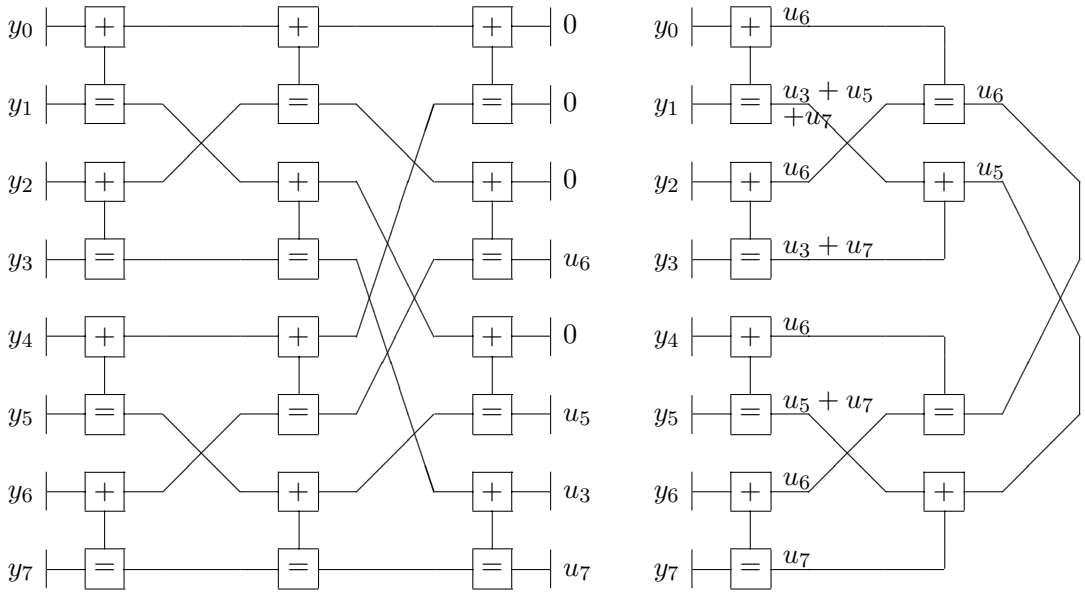

Figure 10. (a) Normal graph of $(8, 4, 4)$ RM code. (b) Equivalent realization.

In Figure 10(b), all state variables are binary and all constraint codes are simple $(3, 2, 2)$ parity-check constraints or $(3, 1, 3)$ repetition constraints. It is believed (but not proved) that this realization is the most efficient possible realization for the $(8, 4, 4)$ code in this sense. However, Figure 10(b) has cycles.

It is easy to see how the cycle-free graph of Figures 7(a) (as well as 7(b), or a minimal four-section, four-state trellis) may be obtained by agglomerating subgraphs of Figure 10(b). Such a graph is depicted in Figure 11. The code symbols are partitioned into four 2-tuples. A state space of dimension 2 connects the two halves of a codeword (meeting the cut-set bound). Two constraint codes of length 6 and dimension 3 determine the possible combinations of symbol 4-tuples and state 2-tuples in each half of the code.

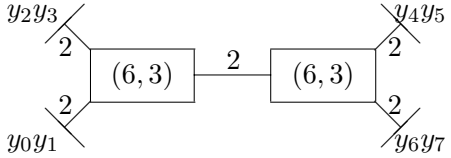

Figure 11. Tree-structured realization of $(8, 4, 4)$ RM code.

Similarly, we may realize any Reed-Muller code $\mathrm{RM}(r,m)$ in any of these styles. By starting with a Hadamard transform realization as in Figure 10(a) and reducing it as in Figure 10(b), we can obtain a realization in which all state variables are binary and all constraint codes are simple $(3,2,2)$ parity-check constraints or $(3,1,3)$ repetition constraints; however, such a realization will generally have cycles. By agglomerating variables, we can obtain a tree-structured, cycle-free realization as in Figure 11 which reflects the $|u|u+v|$ iterative RM code construction.

Exercise 1. (Realizations of repetition and SPC codes)

Show that a reduced Hadamard transform realization of a repetition code $\mathrm{RM}(0,m)$ or a single-parity-check code $\mathrm{RM}(m-1,m)$ is a cycle-free tree-structured realization with a minimum number of $(3,1,3)$ repetition constraints or $(3,2,2)$ parity-check constraints, respectively, and furthermore with minimum diameter (distance between any two code symbols in the tree). Show that these two realizations are duals; *i.e.,* one is obtained from the other via interchange of $(3,2,2)$ constraints and $(3,1,3)$ constraints.

Exercise 2. (Dual realizations of RM codes)

Show that in general a Hadamard transform (HT) realization of any Reed-Muller code $\mathrm{RM}(r,m)$ is the dual of the HT realization of the dual code $\mathrm{RM}(m-r-1,m)$; *i.e.,* one is obtained from the other via interchange of $(3,2,2)$ constraints and $(3,1,3)$ constraints.

Exercise 3. (General tree-structured realizations of RM codes)

Show that there exists a tree-structured realization of $\mathrm{RM}(r,m)$ of the following form:

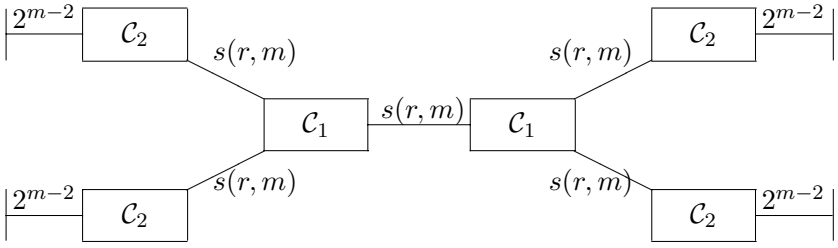

Figure 12. Tree-structured realization of $\mathrm{RM}(r,m)$.

Show that $s(r,m) = \dim \mathrm{RM}(r,m-1) - \dim \mathrm{RM}(r-1,m-1)$ (see Exercise 1 of Chapter 10). Show that the cut-set bound is met everywhere. Finally, show that

$$\begin{aligned} \dim \mathcal{C}_2 &= \dim \mathrm{RM}(r,m-2); \\ \dim \mathcal{C}_1 &= \dim \mathrm{RM}(r,m-1) - 2 \dim \mathrm{RM}(r-2,m-2) = t(r,m), \end{aligned}$$

where $t(r,m)$ is the branch complexity of $\mathrm{RM}(r,m)$ (compare Table 1 of Chapter 6). For example, there exists a tree-structured realization of the $(32,16,8)$ RM code as follows:

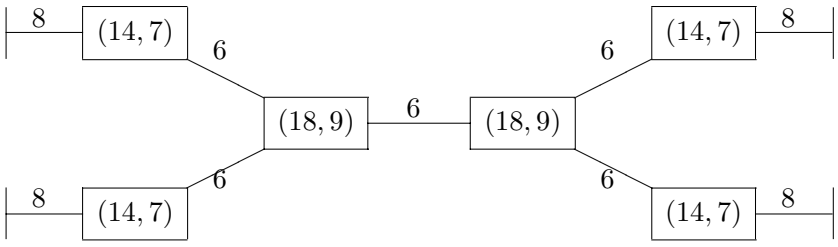

Figure 13. Tree-structured realization of $(32,16,8)$ RM ocde.

11.4 Appendix. Classes of graphical realizations

There are various classes of graphical realizations that can be used for general linear behavioral realizations. Here we will briefly discuss factor graphs, Markov graphs,, and block diagrams.

11.4.1 Factor graphs

A factor graph represents a global function of a set of variables (both internal and external) that factors into a product of local functions defined on subsets of the variables.

The indicator function $\Phi_{\mathfrak{B}}(\mathbf{y}, \mathbf{s})$ of a behavior $\mathfrak{B}$ is a $\{0, 1\}$-valued function of external variables $\mathbf{y}$ and internal variables $\mathbf{s}$ that equals 1 for valid trajectories $(\mathbf{y}, \mathbf{s})$ and equals 0 otherwise. If a trajectory $(\mathbf{y}, \mathbf{s})$ is valid whenever its components lie in a set of local constraint codes $\{\mathcal{C}_k, k \in \mathcal{K}\}$, then the global indicator function $\Phi_{\mathfrak{B}}$ is the product of local indicator functions $\{\Phi_{\mathcal{C}_k}, k \in \mathcal{K}\}$. Thus a behavioral realization may be represented by a factor graph.

A Tanner-type factor graph is an undirected bipartite graph in which variables are represented by one type of vertex (with internal and external variables denoted differently), and functions are represented by a different type of vertex. A Tanner graph of a behavioral realization may be interpreted as a Tanner-type factor graph simply by regarding the constraint vertices as representatives of constraint indicator functions. Similarly, a normal (Forney-type) factor graph is an undirected graph in which internal variables are represented by edges, external variables are represented by dongles, and functions are represented by vertices; in the same way a normal graph of a behavioral realization may be interpreted as a normal factor graph.

In the following chapters, we will be interested in global probability functions that factor into a product of local probability functions; then factor graphs become very useful.

11.4.2 Markov graphs

Markov graphs are often used in statistical physics and statistical inference to represent global probability distributions that factor into a product of local distributions.

A *Markov graph* (Markov random field) is an undirected graph in which variables are represented by vertices, and a constraint or function is represented by an edge (if it has degree 2), or by a hyperedge (if it has degree greater than 2). Moreover, a hyperedge is usually represented by a *clique*, *i.e.,* a set of ordinary edges between every pair of variables incident on the hyperedge. (This style of graph representation sometimes generates inadvertent cliques.)

Markov graphs are particularly nice when the degrees of all constraints are 2 or less. Such a representation is called a *pairwise* Markov graph. We may then represent constraints by ordinary edges. Pairwise constraints often arise naturally in physical models.

Figure 14 shows how any Tanner graph (or Tanner-type factor graph) may be transformed into a pairwise Markov realization by a simple conversion. Here each constraint code has been replaced by a state "supervariable" whose alphabet is the set of all codewords in the constraint code. Each edge then represents the constraint that the associated ordinary variable must be equal to the corresponding component of the supervariable.

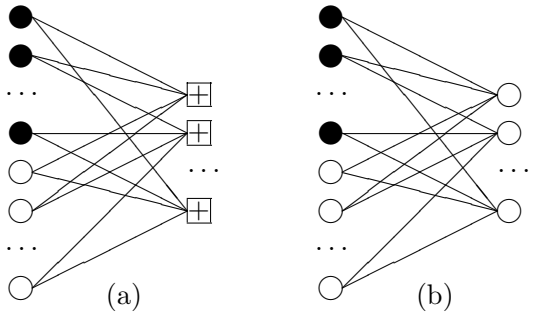

Figure 14. (a) Tanner graph. (b) Equivalent pairwise Markov graph.

For example, suppose the constraint code has degree 3 and constrains three incident variables (y_1, y_2, y_3) to satisfy the parity check $y_1 + y_2 + y_3 = 0$; *i.e.,* the constraint code is a $(3, 2, 2)$ code with four codewords, namely $\{000, 110, 101, 011\}$. We then define a supervariable y_{123} to have these codewords as its alphabet, and constrain y_1 to equal the first component of y_{123}, etc.

11.4.3 Block diagrams and directed normal graphs

Conventional block diagrams may often be regarded as normal graphs, with the vertices ("blocks") representing constraints, and the edges labeled by internal or external variables.

However, one difference is that the blocks usually represent input-output (causal) relationships, so a block diagram is usually a directed graph in which the edges are also labelled with arrows, indicating the direction of causality. In this respect block diagrams resemble Bayesian networks, which are directed acyclic graphs representing probabilistic cause-and-effect models.

This style of graphical model can sometimes be superimposed on a normal graph, as follows. If a constraint code is a linear (n, k) code and has an information set of size k, then the corresponding k symbols may be regarded as "inputs" to the constraint, and the remaining $n - k$ symbols as "outputs" determined by the inputs. Arrows may be drawn on the edges to represent such input-output relationships. If arrows can be drawn consistently on all edges in this way, then a normal graph may be converted to a directed normal graph (block diagram).

For example, Figure 15 shows how a parity-check realization for the $(8, 4, 4)$ code (Figure 3(b)) may be converted to directed normal graph form. This could be useful if, for example, we wanted to use such a graph to implement an encoder. However, this example is a bit misleading, as parity-check realizations cannot always be converted to encoders in this way.

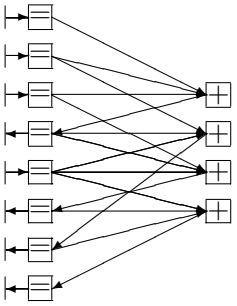

Figure 15. Conversion of parity-check realization of $(8, 4, 4)$ code to directed normal graph representing an encoder.