Chapter 2 Fundamental Graph Theory

2.1 Introduction to Directed Acyclic Graph, Directed Markov Field and Bayesian Network

In this subsection, we will introduce some important and necessary notions of directed acyclic graph, directed Markov field and Bayesian belief network, or simply described as Bayesian network for short, through several intuitive but heuristic examples, which are very important to characterize the ordinary linear error-correcting codes, such as block and convolutional codes, and some well know concatenated codes, such as low-density parity-check (LDPC) codes, parallel concatenated (turbo) codes, serially concatenated codes, etc.

Let us first consider a simple graph shown in Fig. 2.1 with a finite set of variables V= {V_1, V_2, …, V_6} with one-to-one correspondence to a set of vertices $\overline{\boldsymbol{V}}=\{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_6\}$, and each of the vertices connects with other vertices according to the set of directed edges E. A directed graph is represented by G=($\overline{\boldsymbol{V}}$, E).

Definition 2.1: A *directed acyclic graph* (DAG) [22][23] is a finite directed graph where there are no graph cycles when the edge directions are followed (or no directed circles). A DAG is called a tree if it contains no loops. □

Example 2.1: According to the definition 2.1, the graph depicted by Fig. 2.1 is certainly a DAG with seven edges. Meanwhile, we should bear in mind that the phase “loop”, denoting a cycle in the underlying undirected graph, is completely different from the directed circle. The girth of a graph is the length of its shortest cycle [31]. For instance, $\bar{v}_1 \to \bar{v}_5 \to \bar{v}_6 \to \bar{v}_3 \to \bar{v}_1$ is one of the loops and girth is 4 in the example. Thus, it is not a tree. However, if we cancel the two directed edges shown as indicated by the bold dashed lines, then the DAG immediately becomes a tree since all the loops are deleted in the DAG. The tree is also an important notion that will lead to a broad category of error-correction codes applicable for decoding by strict belief propagation algorithm. □

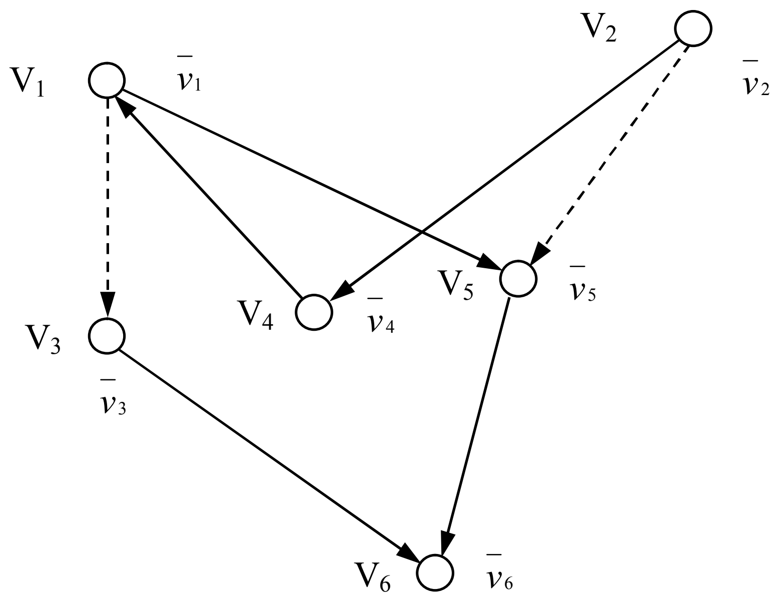


Fig. 2.1. An illustrative example of a simple graph with six variables connected by the directed edges and with no directed circles.

Further examine the vertex $\bar{v}_5$ in the example, there are two directed edges pointed away from the vertices $\bar{v}_1$ and $\bar{v}_2$, then $\bar{v}_5$ is called a "child" of $\bar{v}_1$ and $\bar{v}_2$, in return, $\bar{v}_1$ and $\bar{v}_2$ are called "parents" of $\bar{v}_5$. If the set of parents of any vertex $\bar{v}_i$ (i=1, 2, …, 6) is denoted by $Pa(\bar{v}_i)$, the directed interconnections between the vertices of the graph depicted by Fig. 2.1 can be explicitly described as follows:

$$Pa(\bar{v}_1)=\{\bar{v}_4\} \qquad Pa(\bar{v}_2)=\phi \qquad Pa(\bar{v}_3)=\{\bar{v}_1\} \tag{2-1a}$$

$$Pa(\bar{v}_4)=\{\bar{v}_2\} \qquad Pa(\bar{v}_5)=\{\bar{v}_1,\ \bar{v}_2\} \qquad Pa(\bar{v}_6)=\{\bar{v}_3,\ \bar{v}_5\} \tag{2-1b}$$

Surprisingly and interestingly, one "child" may have no "father", such as $\bar{v}_2$. However, it is exactly true for the mathematical definition, and also completely necessary to tackle the crucial problem of extrinsic information flowing through the edges of the tree codes while belief propagation algorithm is employed for decoding. These extremely important details will be explained lucidly in the subsequence of this chapter. Generically, if a graph G is a DAG, and **V** is a set of N random variables in one-to-one correspondence with the vertices set $\{\bar{v}_1,\ \bar{v}_2,\ \ldots,\ \bar{v}_N\}$ of G, the joint probability density function $p(\bar{v}_1,\ \bar{v}_2,\ldots,\ \bar{v}_N)$ is said to factor according to G if

$$p\,(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_N)=p(\bar{v}_1)\ p(\bar{v}_2|\bar{v}_1)\ p(\bar{v}_3|\bar{v}_1,\bar{v}_2)\times$$

$$\ldots\ldots\times p(\bar{v}_N|\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_{N-1})=\prod_{i=1}^{N}P(\bar{v}_i|Pa(\bar{v}_i)) \tag{2-2}$$

For the DAG in Fig. 12.1, the joint probability density must satisfy the condition as follows:

$$p(\bar{v}_1,\bar{v}_2,\ldots,\bar{v}_6)=\prod_{i=1}^{6}p(\bar{v}_i|Pa(\bar{v}_i)) \tag{2-3}$$

Usually, the evaluation of conditional probability of a variable needs all the observations acquired so far, as expressed by the first equation in right-hand side in (2-2). However, if a “child” is only concerned with its own “parents”, i.e., it has nothing to do with the “parents” of other “children”, or in terms of standard phase from the probability theory, any variable of the graph only depends on the variables in the graph based on the directed edges that point to, and thus any vertex in the graph definitely relates to a Markov random variable of lower order. The joint probability density function will be greatly simplified by fully making use of this potential advantage and indicated by the second equation of right-side of (2-2). Actually, this fact is not new to the reader, as for the convolutional codes the future events are not influenced by the past observations and inputs if the current state of the code is known. Now we can give a mathematical definition of the Bayesian network as:

Definition 2.2: A set of random variables **V**, whose joint density function agrees to (2-2) according to a given DAG, is called a *directed Markov filed* [29]. A DAG, together with the associated random variables set **V**, is called a Bayesian belief network, or simply Bayesian network for short [23][34][29][30]. □

The set of variables **V** is an ordinary Markov chain if G is a directed chain. Usually without causing any confusion, we will only mark random variables on the Bayesian network in most situations for the sake of analytical convenience.

The idea hidden behind so-called Bayesian network is to exploit any “partial or local dependencies”, which may exist among the variables, and thus to simplify the problem of probabilistic inference in the DAG.

Let the J be a subset of $\mathbf{I}_N=\{1, 2, \ldots, N\}$, i.e., $J\subseteq\mathbf{I}_N$, such that, for all $j\in J$, the random variable V_j is known (or observed) to have a particular value, saying v_j. A set of all the observations is described as

$$\varepsilon=\{V_j=v_j \mid j\in J\} \qquad (12.28) \qquad (2\text{-}4)$$

The fundamental problem of probabilistic interference is to compute the *a posterior* or conditional probabilities $p(V_i|\varepsilon)$ for all $i\notin J$ based on the respective observations.

For the probabilistic decoding of some error-correction codes that can be effectively represented by directed acyclic graphs, such as turbo codes, serially concatenated codes, LDPC codes and generalized tree codes introduced in the later parts of this chapter, the ultimate goal of a decoder is to evaluate and update the *a posterior* probabilities (APP’s) when the message flows through the graph of the codes, such as like SISO APP module for iterative decoding of parallel and serially concatenated codes [37]. This should be one of the most important applications of probabilistic interference in communication and information theory.

Here we should, at least, intuitively explain the reason why the condition of “no directed circles” is particularly emphasized on the definition 2.2 for any DAG. Suppose a directed edge rooted from the variable V_1 be added and pointed to the variable V_2 in the DAG sketched in Fig. 2.1, thus a directional cycle immediately follows, which is illustrated by bold dashed lines in Fig. 2.2, i.e., $\bar{v}_1 \to \bar{v}_2 \to \bar{v}_4 \to \bar{v}_1$. If a "parent", saying $\bar{v}_1$, sends a message to his "child" $\bar{v}_2$, who conveys the identical message to the child's "child" $\bar{v}_4$, and eventually the same message is fed back to $\bar{v}_1$ (now updated as the "grandparent"). For the variable $\bar{v}_1$, it obviously gets the message containing no information at all or the information with zero entropy. Definitely, such a directed circle partially or completely infringes the process of probabilistic inference, which enables any "child" evaluates the conditional probability based on its "parents" when the variable sends/receives the message along the directed edges of the graph. Thus, this is the reason why such a scenario must be strictly prohibited in the DAG.

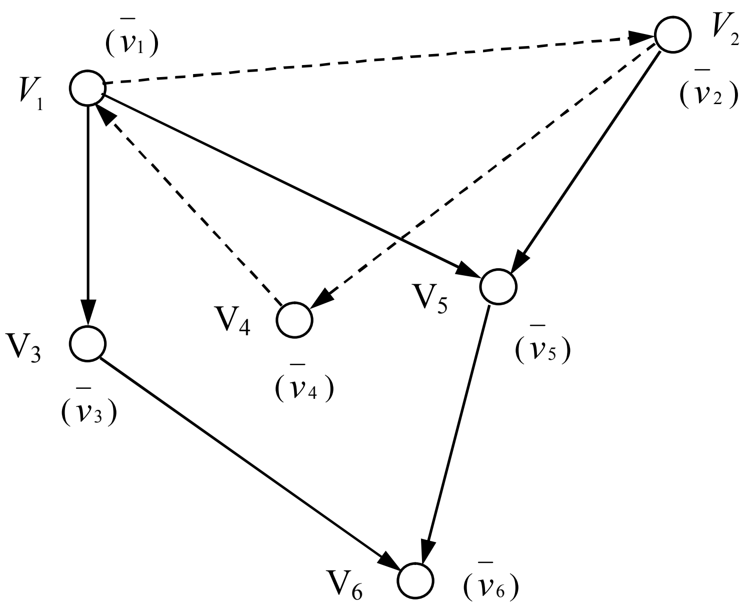
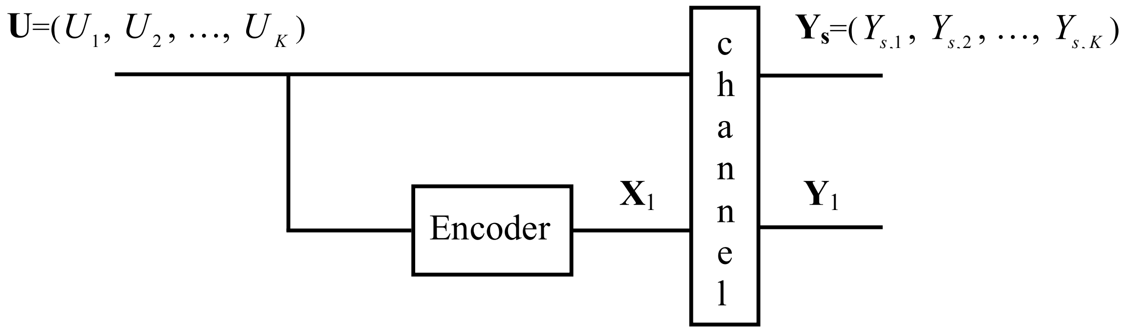
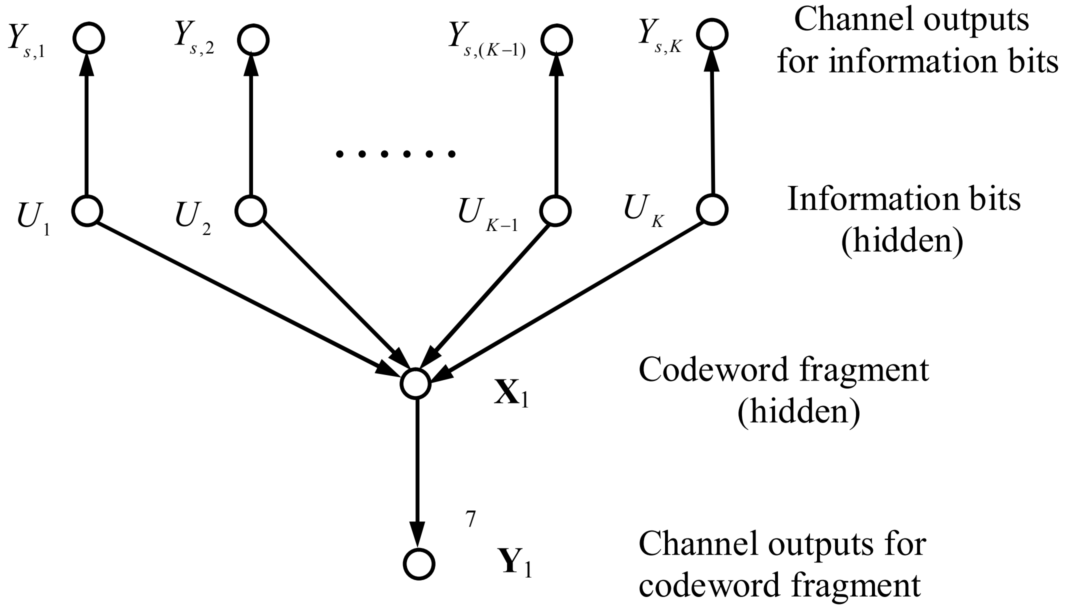


Fig. 2.2. An illustrative example of a graph with six variables connected by the directed edges and with a directed circle plotted by the dashed lines.

The generic coding framework and Bayesian network expression are shown in Fig. 2.3(a) and (b), respectively. From the decoder's viewpoint, the observed channel outputs $Y_{s,i}$ (i=1, 2, …, K) are probabilistic functions of the information bits U_i. Similarly, the observed codeword fragment $\mathbf{Y}_1$ is a probabilistic function of the codeword $\mathbf{X}_1$, which in turn is a deterministic function of the input bits. Therefore, the decoder's problem is to infer the values of U_i based on the observation variables ($Y_{s,1}$, $Y_{s,2}$, …, $Y_{s,K}$) and $\mathbf{Y}_1$. Note that only observations are actually visible.


$\mathbf{U}$=(U_1, U_2, …, U_K)
Encoder
$\mathbf{X}_1$
c h a n n e l
$\mathbf{Y_s}$=($Y_{s,1}$, $Y_{s,2}$, …, $Y_{s,K}$)
$\mathbf{Y}_1$

(a)


$Y_{s,1}$
$Y_{s,2}$
$Y_{s,(K-1)}$
$Y_{s,K}$
Channel outputs for information bits
U_1
U_2
U_{K-1}
U_K
Information bits (hidden)
$\mathbf{X}_1$
Codeword fragment (hidden)
$\mathbf{Y}_1$
Channel outputs for codeword fragment

2.2 Bayesian Networks for Some Typical Error-Correcting Codes

In order to provide an insight view for the reader, we now present several interesting and heuristic examples to illustrate how a Bayesian network can effectively characterize a linear code from the viewpoint of encoder/receiver. Let us first investigate a simple example, i.e., best-known binary Hamming code.

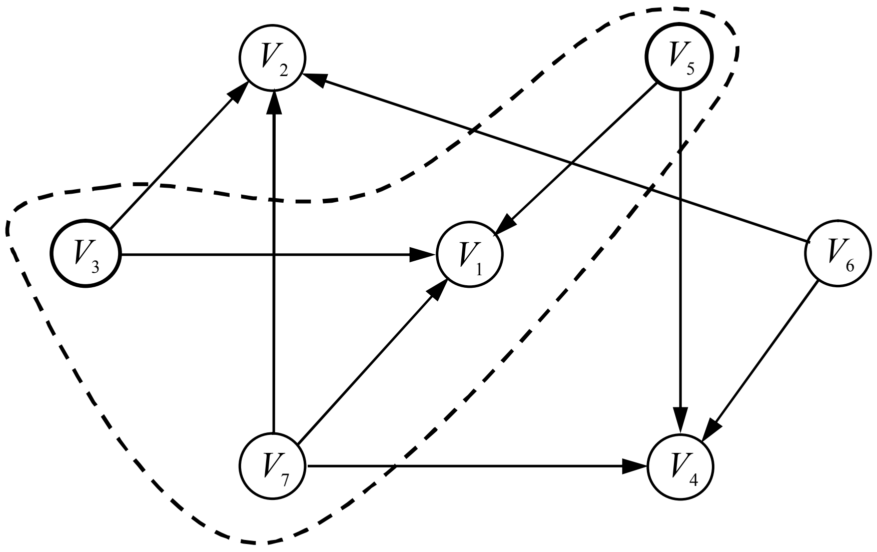


Fig. 2.4 Bayesian network for a non-systematic binary (7, 4) Hamming code.

Example 2.2 *Binary Hamming Code*: The parity check matrix of a nonsystematic binary (7, 4) Hamming code is

$$H_{3\times7}=\begin{bmatrix}0&0&0&1&1&1&1\\0&1&1&0&0&1&1\\1&0&1&0&1&0&1\end{bmatrix} \qquad (2\text{-}5)$$

Suppose the codewords, conveying the information bits, be transmitted through the binary symmetry channel (BSC), and the received bits corrupted by the channel noise be the binary random variables over GF(2), which are associated with the vertices in the graph of the code. The directed edges connected between the variables are specified by the parity check matrix (2.5). Consequently, the graph representation of the Hamming code is depicted in Fig. 2.4, where the random variables V_1, V_2, …, V_7 in the graph stand for a block of received bits. The group of variables V_1, V_3, V_5, V_7, circled by the dashed cure, forms a parity check that corresponds to the third parity check of matrix $H_{3\times7}$. The same scenario can be found for the other two groups of variables $\{V_2, V_3, V_6, V_7\}$ and $\{V_4, V_5, V_6\ V_7\}$ that relate to the first and second parity checks of the matrix. Evidently, the graph of the (7, 4) Hamming code is a DAG, but not a tree since there exists loops (undirected circle) one of which is $V_1 \to V_5 \to V_4 \to V_7 \to V_1$.

Since a block code is completely determined by either the generator matrix or the equivalent parity-check matrix, any received variable in a block only depends on the variable(s) connected by the directed edge(s). Thus, the joint probability density function is

$$p(V_1 = v_1, V_2 = v_2, \ldots, V_7 = v_7) = p(V_1 = v_1 | V_3 = v_3, V_5 = v_5, V_7 = v_7)$$
$$\times p(V_2 = v_2 | V_3 = v_3, V_6 = v_6, V_7 = v_7)\ p(V_3 = v_3)\ p(V_4 = v_4 | V_5 = v_5, V_6 = v_6, V_7 = v_7)$$
$$\times p(V_5 = v_5)\ p(V_6 = v_6)\ p(V_7 = v_7) \qquad (12.30) \qquad (2\text{-}6)$$

In this sense, the graph further becomes a Bayesian network. □

The aforementioned Bayesian network also can be used to depict the encoding process of the Hamming code. Apart from the random bits sent from the information source, the encoder itself is a deterministic mechanism. The conditional probability is simplified to a binary case with variables interpreted as codeword bits. For $V_1 = v_1$ we have

$$\Pr(V_1 = 0 | V_3 = v_3, V_5 = v_5, V_7 = v_7) = 1 \text{ if } v_3,\ v_5 \text{ and } v_7 \text{ have even parities} \qquad (2\text{-}7a)$$

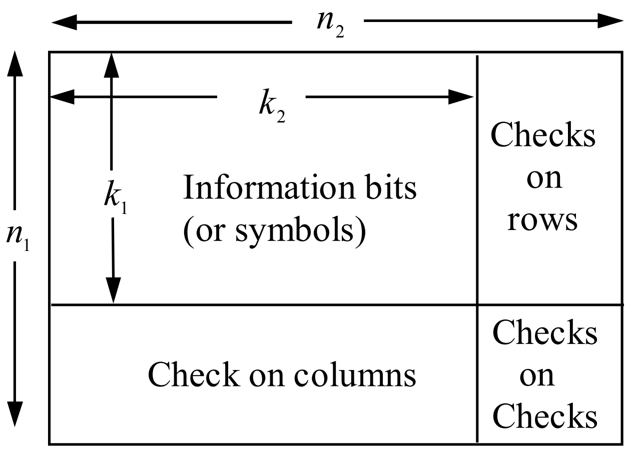
$$\Pr(V_1 = 1 | V_3 = v_3, V_5 = v_5, V_7 = v_7) = 1 \text{ if } v_3,\ v_5 \text{ and } v_7 \text{ have odd parities} \qquad (2\text{-}7b)$$

In other words, if the three encoded bits v_3, v_5 and v_7 are known, e.g., v_3=0, v_5=1 and v_7=1, then v_1 must choose 0. If v_3=1, v_5=0 and v_7=0, then v_1 must be 1. The remainder of the codeword bits can be similarly specified through the other parity checks as illustrated by Fig.12.4.

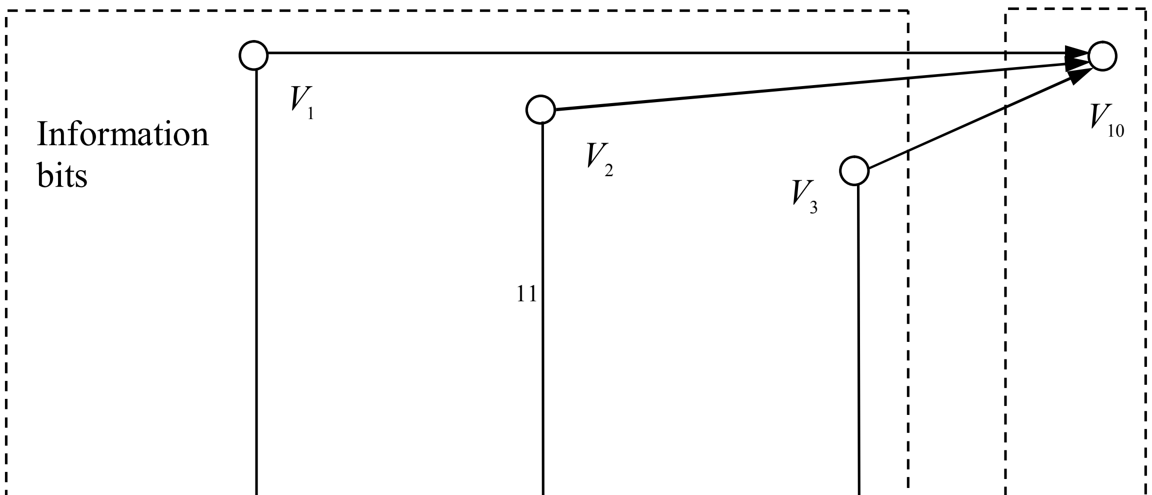
Evidently, a (7, 4) binary Hamming code can be fully characterized by the related Bayesian network. Thus, it may shed a promising light that people can effectively investigate more complicated error-correcting codes by means of graph theory and related analytical tools.

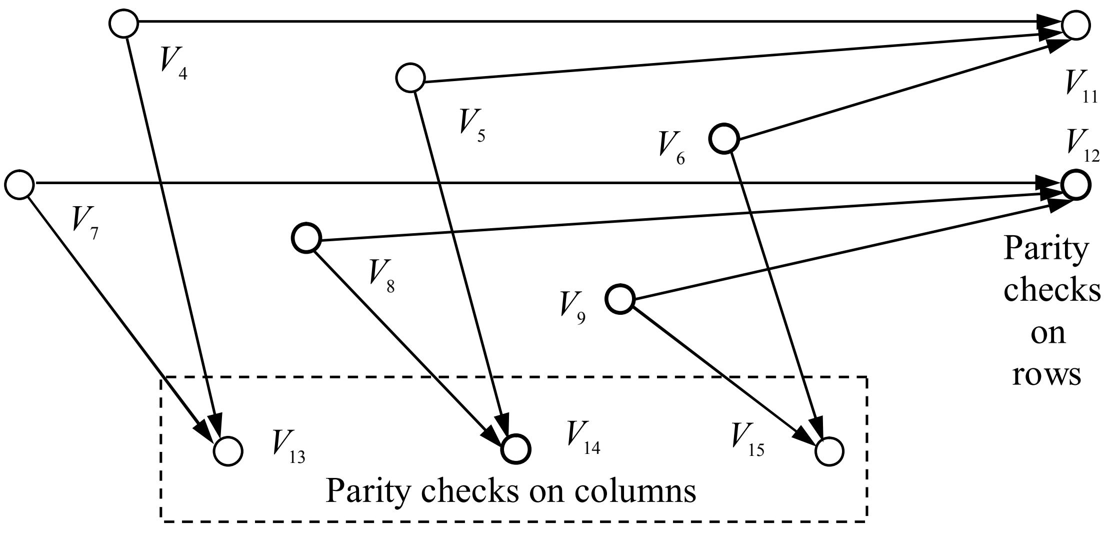
Then, we will concentrate on some more complicated examples, i.e., product codes, turbo codes, serially concatenated codes that have previously studied, and the LDPC codes, which attract our much interest in this book.

Example 2.3 Consider a generic product code, shown in Fig. 5(a), and Bayesian network description for a simple binary (15, 9) product code, illustrated by Fig. 5(b), which is subject to the conditions as $n_1=n_2=4$, $k_1=k_2=3$, and no "checks on checks". Obviously, this example demonstrates the explicit construction of a product code, i.e., a single parity bit checks the parity of each row and column of the information bit array.



(a)



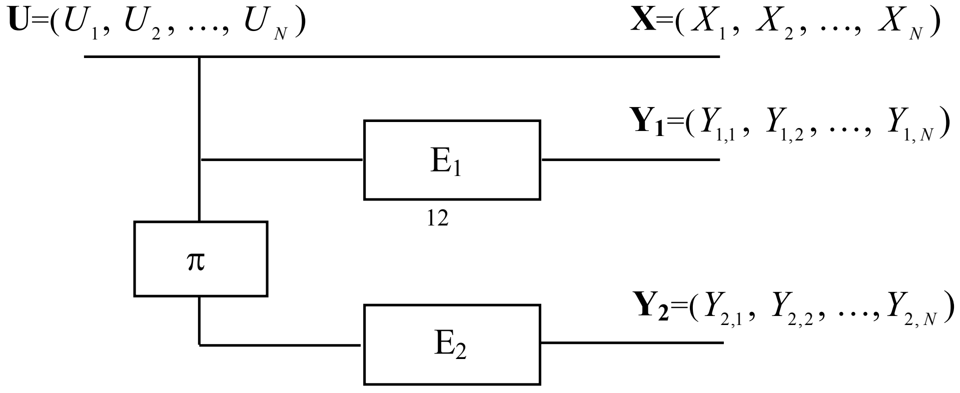


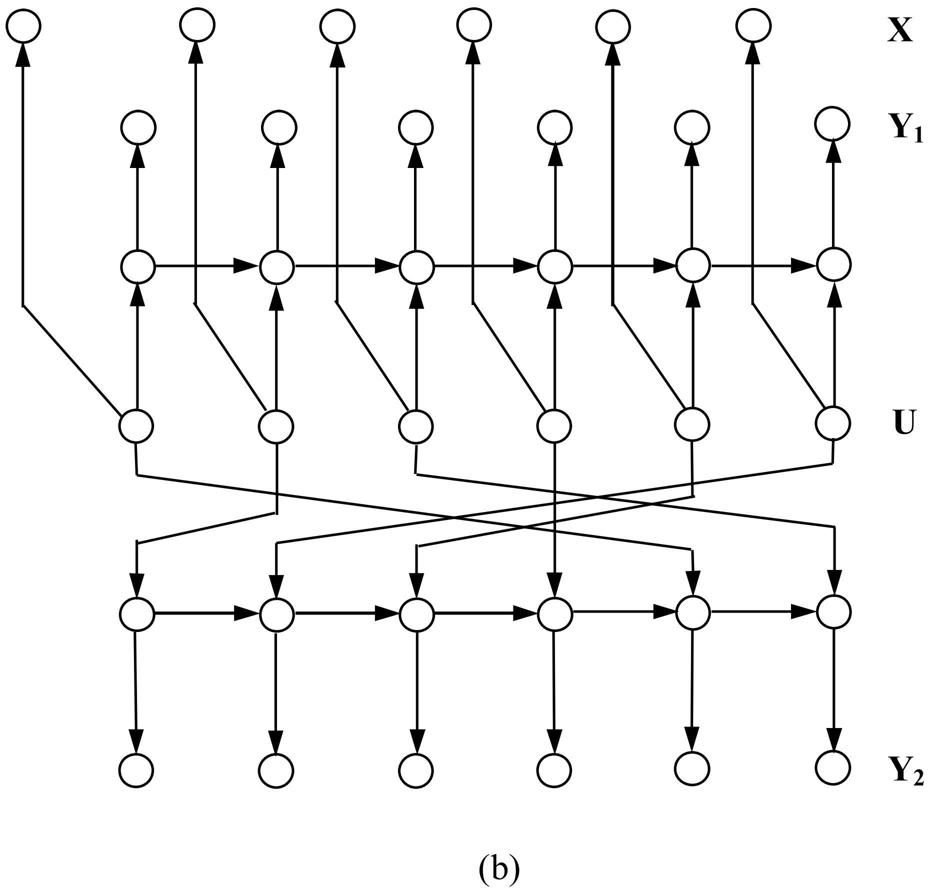
(b)

Fig. 2.5 The illustrative diagrams of (a) a generic product code, (b) the Bayesian network representation for a simple binary (15, 9) product code.

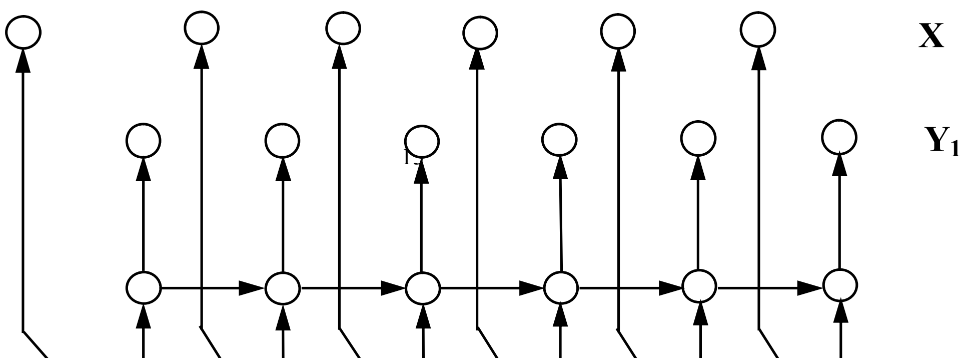
Note that DAG of the (15, 9) product code is not a tree as one of the loops is $V_1 \to V_{10} \to V_3 \to V_{15} \to V_9 \to V_{12} \to V_7 \to V_{13} \to V_1$. For the same reason as Example 2.2, the variables of the above Bayesian network can also be interpreted as the received symbols for the decoder, which are tackled properly based on the parity-check relationships made by the encoder. □

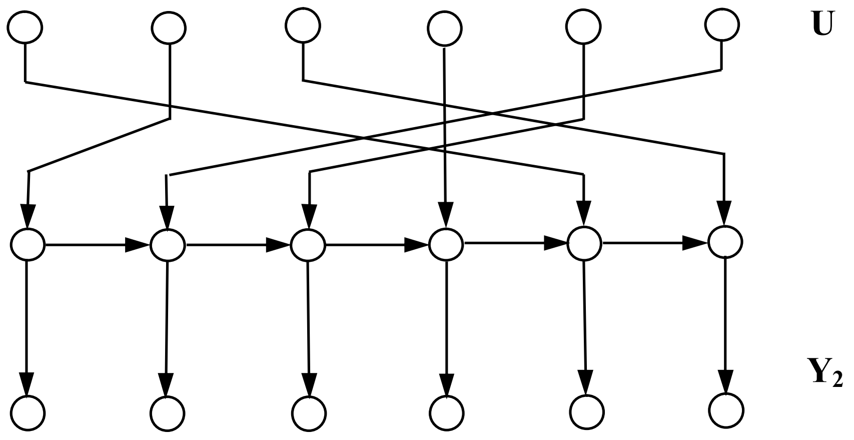
Example 2.4 *Turbo Code*: The illustrative diagrams of a rate-1/3 turbo encoder, and related Bayesian network descriptions for the overall code and two constituent codes are shown by Fig. 2.6(a)~(d), respectively.


$\mathbf{U}=(U_1, U_2, \dots, U_N)$
$\mathbf{X}=(X_1, X_2, \dots, X_N)$
$\mathbf{Y_1}=(Y_{1,1}, Y_{1,2}, \dots, Y_{1,N})$
E_1
π
$\mathbf{Y_2}=(Y_{2,1}, Y_{2,2}, \dots, Y_{2,N})$
E_2



(b)



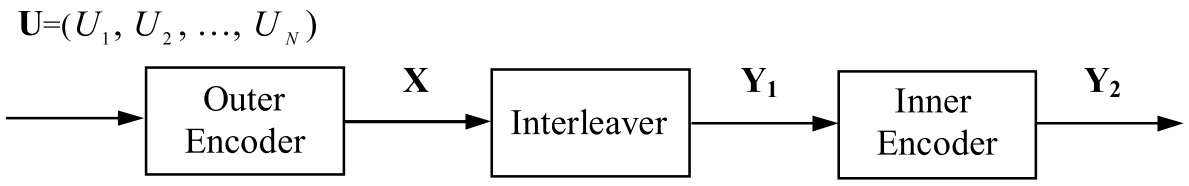


(d)

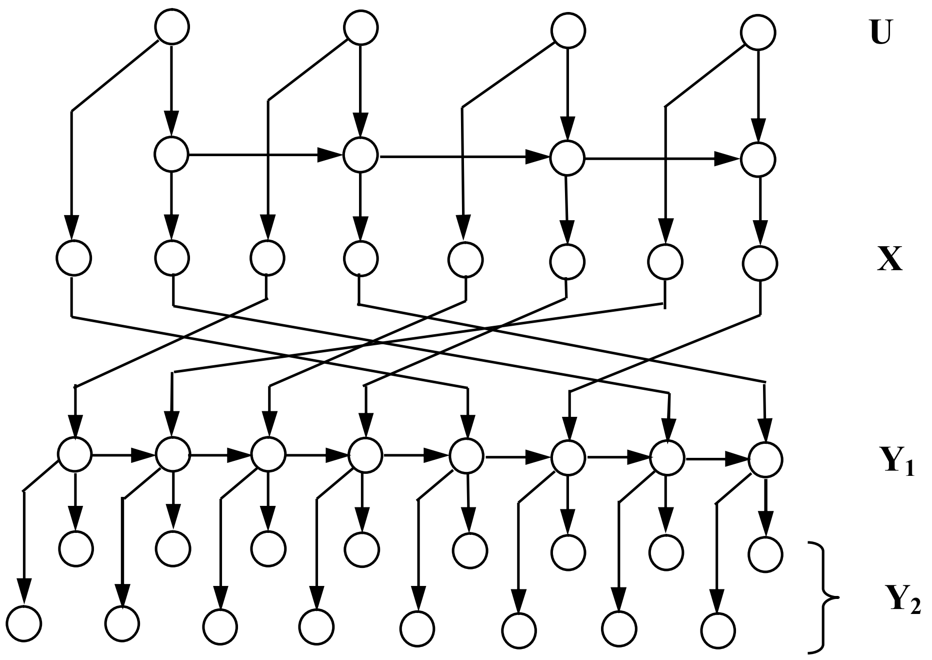
Fig. 2.6 The illustrative diagrams of (a) a rate-1/3 turbo code, associated Bayesian networks for (b) overall turbo code, (c) the first constituent code, (d) the second constituent codes.

For the sake of simplicity for the illustration, we only use a block of six bits (N=6) for the turbo encoder. Of course, there is no turbo code with such a short block length in practice. Graphically, Bayesian networks for an overall turbo code can be equivalently represented by two loop-free networks of the component codes. Note that the variables of the fourth and fifth layer in Fig. 2.6(b) are connected by the directed edges determined by an interleaver. Evidently, it is a loopy Bayesian network for the overall turbo code. □

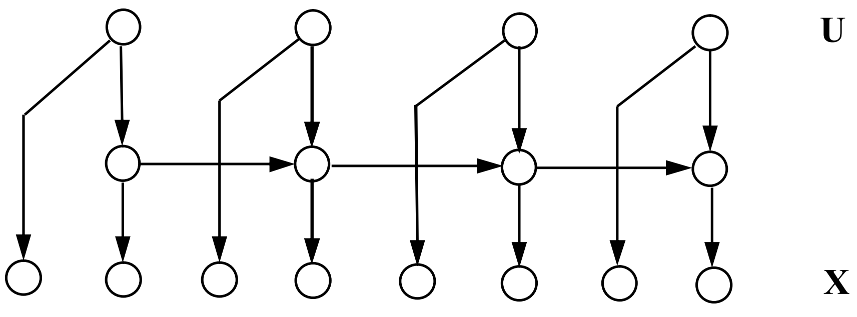
Example 2.5 *Serially Concatenated Convolutional Code*: The illustrative diagrams of a serially concatenated convolutional code (SCCC), and Bayesian networks for a rate-1/4 SCCC and two rate-1/2 outer and inner constituent convolutional codes are shown in Fig. 2.7(a)~(d), respectively.



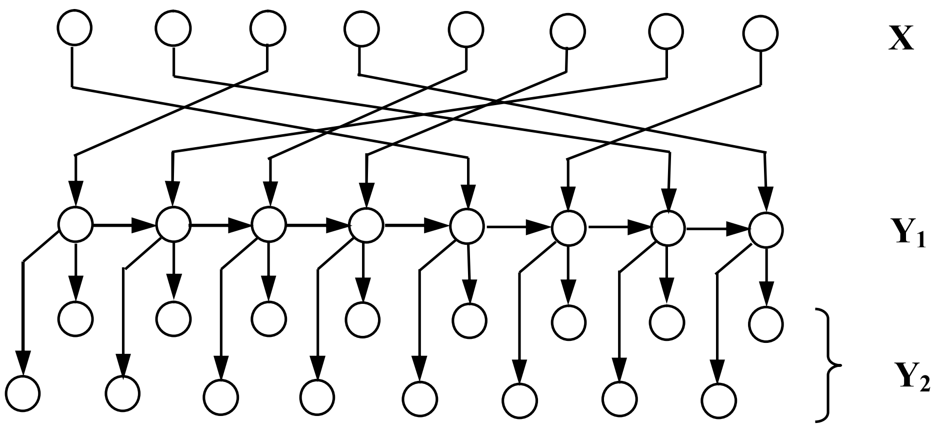
(a)



(b)



(c)



(d)

Fig. 2.7 The illustrative diagrams of (a) a SCCC, Bayesian networks for (b) a rate-1/4 SCCC, (c) a rate-1/2 outer constituent convolutional code, (d) a rate-1/2 inner constituent convolutional code.

For the similar reasons as the turbo codes, the block length of information bits is 4 and thus the length of the output of a SCCC is 16 due to the concatenated two rate-1/2 outer and inner component convolutional codes. The DAG's of the component convolutional codes are trees while the DAG of SCCC is not a tree, which exhibits the same scenario as the turbo code (PCCC) illustrated in Fig. 2.6. Thus, the concatenated codes themselves, both for PCCC and SCCC, are not tree codes. □

The next example is involved with the phenomenal regular low-density parity-check (LDPC) codes, the reader can see this example after the study of the so-called Tanner graph [2][14][15] and the definition of LDPC codes, which appears in the Chapter 3.

Example 2.6 *Low-Density Parity-Check Code*: Consider a regular LDPC code, whose parity-check matrix is defined by (2-8). The block length is 9 with d_v=2 and d_c=3, and its Bayesian network description is shown in Fig. 2.8.

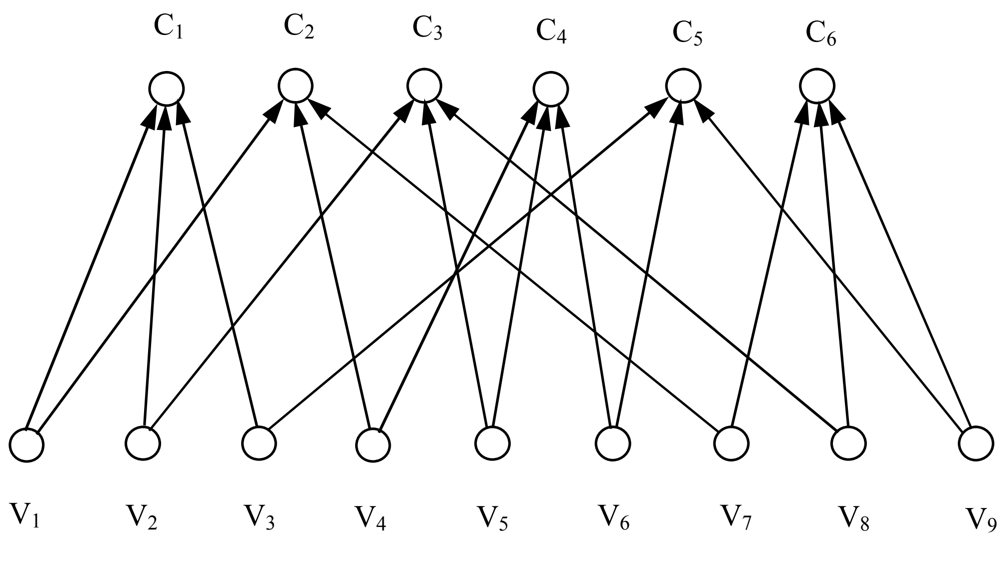


Fig. 2.8 The Bayesian network for a regular LDPC code of block length 12 with d_v=2 and d_c=3.

$$H_{9\times 6}=\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad (2\text{-}8)$$

The check variables and block codeword bits are associated with the first and second layer of the Bayesian network, respectively. Interestingly, it looks like the well-known bipartite graph that is applied to characterize regular LDPC codes. Actually, a bipartite graph [2][14] is also called a Tanner graph that will be introduced in the subsequent chapter. Like the previously mentioned PCCC and SCCC, the DAG of this LDPC code is also not a tree because it contains many loops, one of which, for example, is $C_1 \rightarrow V_2 \rightarrow C_3 \rightarrow V_8 \rightarrow C_6 \rightarrow V_7 \rightarrow C_2 \rightarrow V_1 \rightarrow C_1$. Finally, the interested reader may try to depict the Bayesian network of a regular LDPC code shown in Fig. 4.3. □

2.3 Tanner Graph, Tree Codes and Concatenated Tree Codes

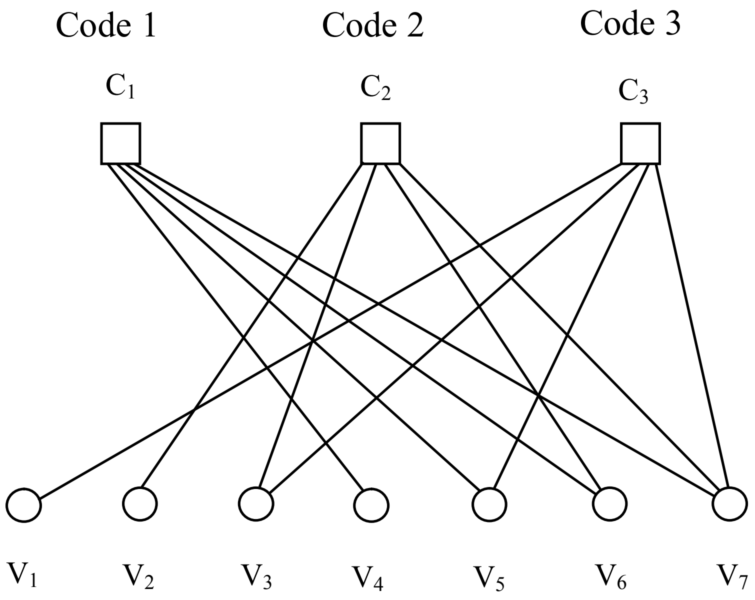
In the last section, we briefly recall the basic knowledge of the graph, DAG, Bayesian network and applications for the descriptions of some well-known error-correcting codes. In the 1980's, Kim and Pear showed that there are efficient distributed (BP) algorithms for solving the general inference problem if the DAG is a tree [22][38]. Moreover, the Pear's belief propagation [5][22] can be defined for an arbitrary DAG which is not necessary a tree, even through there is no guarantee that the algorithm will perform a useful calculation for the loopy DAG.

For most of practical problems of decoding the concatenated codes, especially for LDPC codes, we are more concerned with developing an effective iterative decoder using the BP algorithm since the concepts of DAG and Bayesian network seem to be too broad for our purposes. Therefore, a new pragmatic graphic model is absolutely necessary. However, the previously introduced notions definitely form a fundamental basis for understanding the new model, i.e., Tanner graph, which is explained in this section.

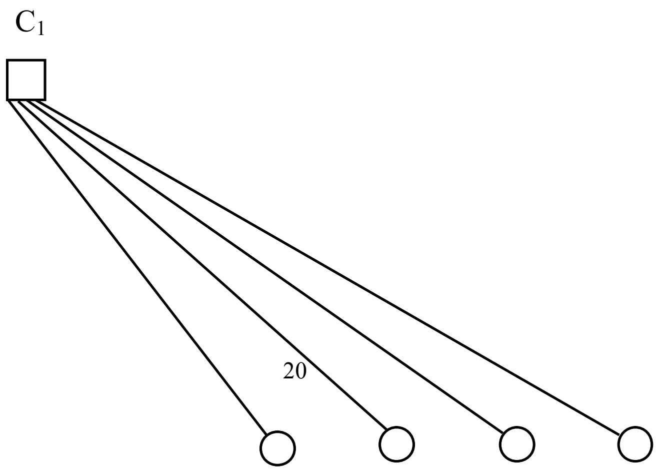
In 1981, Tanner originally proposed a kind of graph [2], whose central idea is to construct the long error-correcting codes by means of one or more shorter error-correcting codes, referred to as component codes. The long code, consisting of the individual shorter component codes, is represented by a bipartite graph, which has been directly adopted to define a regular LDPC code and will be continuously used for the irregular LDPC codes. Consequently, the lower bounds of rate and minimum distance [2][14] of these component codes can be derived more easily due to the shorter length and relatively simple construction. Both the encoder and decoder take advantage of the code's explicit decomposition into the component codes to decompose and simplify the associated computational processes.

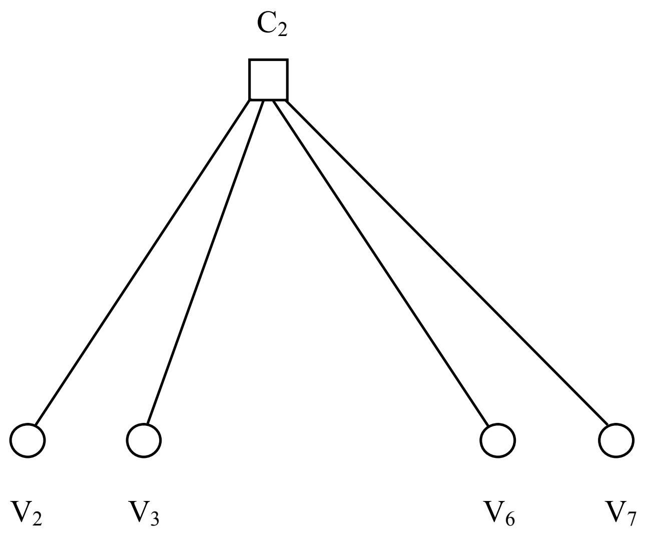
In order to clearly demonstrate the details of the bipartite graph and the related meaningful impacts for the belief propagation algorithm, here we will still explore some well-known simple codes that have been used as intuitive examples, i.e., Hamming code and LDPC code, in the previous sections.

Example 2.7 For the binary (7, 4) Hamming code with parity-check matrix defined by (2-5) and equivalently represented by the Bayesian network as shown in Fig. 2.4, a group of new nodes corresponding to the parity checks are added to the graphs as well as the existing 7-node version. These new graphs are referred to as Tanner graph [2] and redrawn in Fig. 2.9.

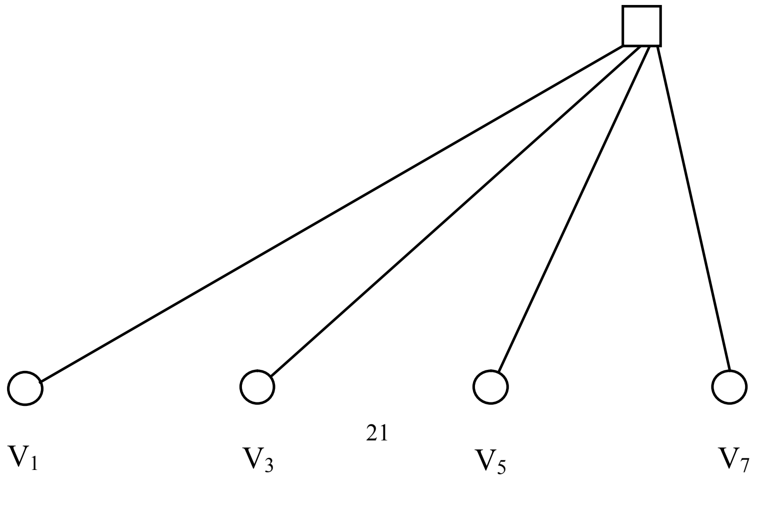


(a)





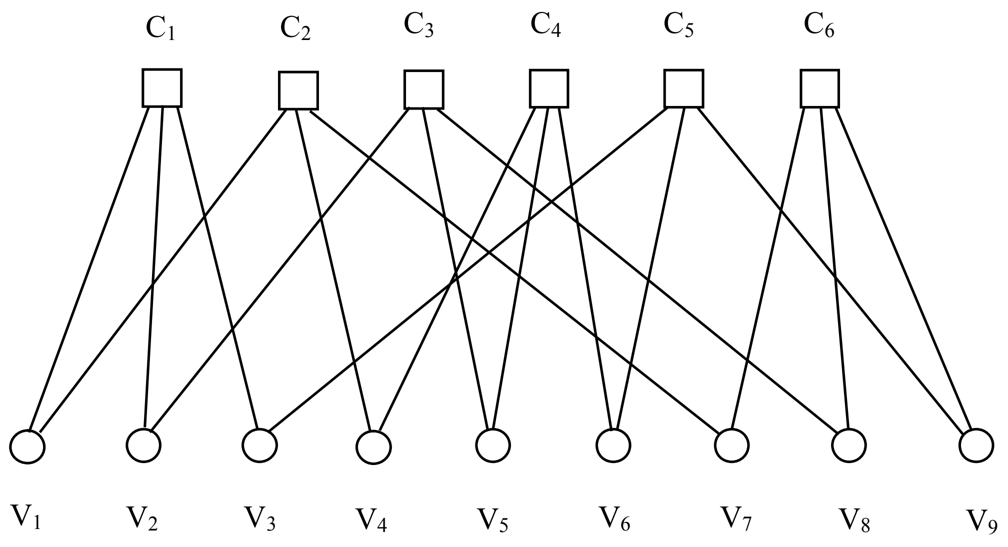
(c) code 2



All the 10 nodes belonging to two distinct categories are placed in two different layers, as shown in Fig. 2.9(a). The 3 nodes in the first layer are the check nodes corresponding to the 3 parity checks defined by the check matrix of the Hamming code. The other 7 nodes in the second layer are the variable nodes related to codeword bits, which are the same as those in the Bayesian network shown in Fig. 2.4. Notice all the edges only link the nodes of different categories and the so-called Tanner graph is an undirected graph.

Let us further examine the check node C_1 and the related variable nodes that are V_4, V_5, V_6 and V_7, respectively. Actually, these four constrained bits, subject to the parity check C_1, can result in a (4, 3) block code denoted by *code* 1 in Fig. 2.9(b). The other two (4, 3) block codes, denoted by *code* 2 and *code* 3 that are associated with the parity checks C_2 and C_3, respectively, can be similarly constructed and shown in Figs. 2.9(c) and (d), respectively. This defines the overall codewords of a (7, 4) Hamming code in an intersection of all the component codes: given 7 bits, which is a valid codeword if and only if (iff) the related bits for each parity check form a codeword of the corresponding component code. It is also straightforward for other block and convolutional codes. □

Example 2.8 Consider a regular LDPC code that is previously illustrated in the example 2.6. The bipartite graph representations of the LDPC and one of a component codes are easily depicted in Fig. 2.10(a) and (b), respectively.



(a)

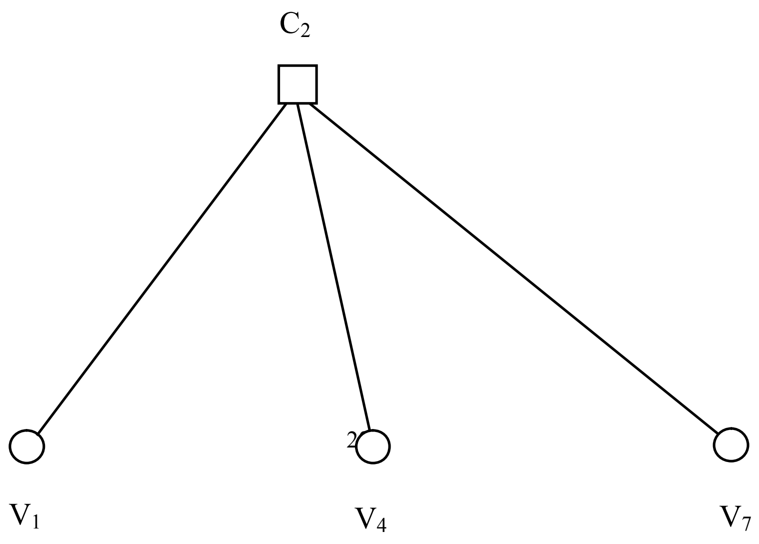


Fig. 2.10 (a) The bipartite graph for a regular LDPC code of block length 9 with d_c=3 and d_v=2. (b) The bipartite graph of a (3, 2) block code related to the second check node.

Similar as the aforementioned (7, 4) Hamming code, the bipartite graph for the LDPC code has 9 variable nodes and 6 check nodes as shown in Fig. 2.10(a). Each of parity checks relates to three bits and all the component code are simple (3, 2) block codes, one of which is illustrated in Fig. 2.10(b) for the second check node. Therefore, such a LDPC code can be also constructed by an intersection of its individual component codes. □

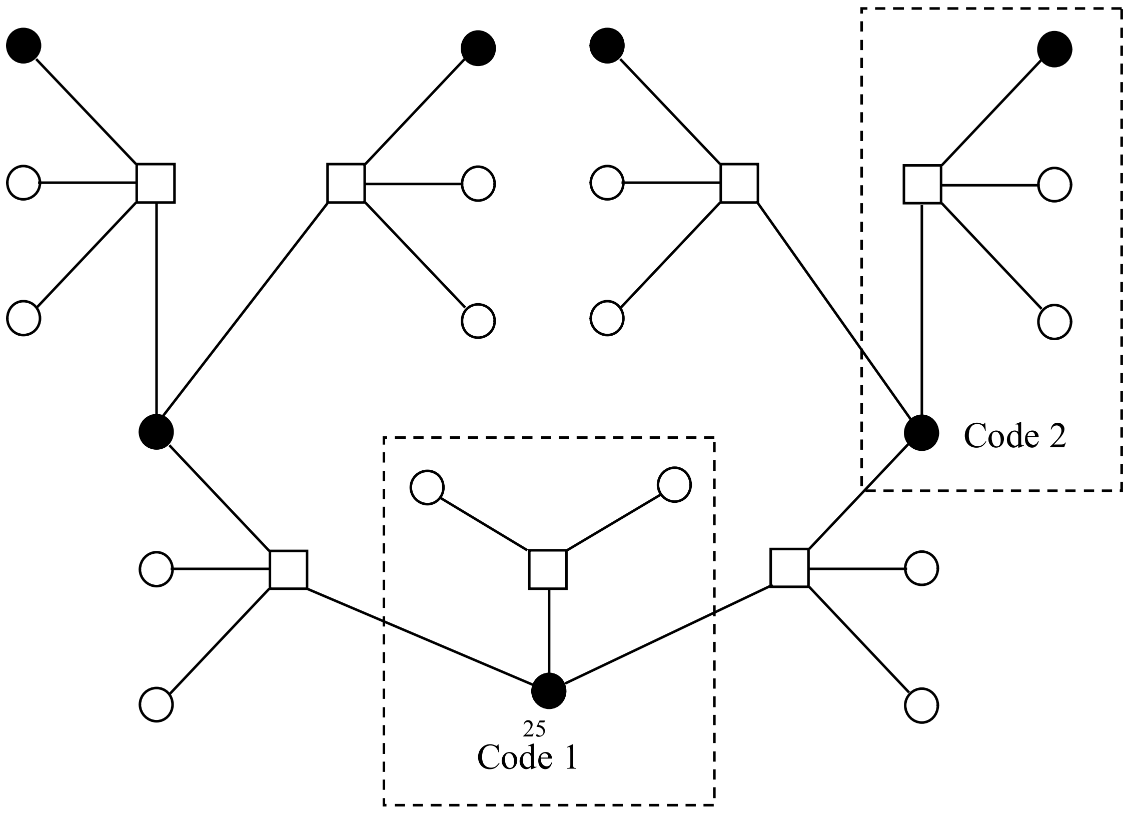
Based on above simple examples, we can clearly observe that bipartite graphs of both Hamming code and regular LDPC codes studied are not tree although all their component codes have loop-free graphs. Furthermore, long codes can be built from a bipartite graph and one or more component codes; a new code is defined explicitly by its decomposition into shorter codes. The interested reader may try to describe the binary (15, 9) product code as shown in Fig. 2.5 by means of Tanner graph.

Finally, it is the appropriate time to give the mathematical definitions for the Tanner graph and tree codes based on the previous intuitive examples.

Definition 2.3 A bipartite graph is an undirected graph in which the nodes can be portioned into two distinct classes. An edge of the graph may connect a node of one class to a node of another class, but there are no edges connecting nodes of the same kind. □

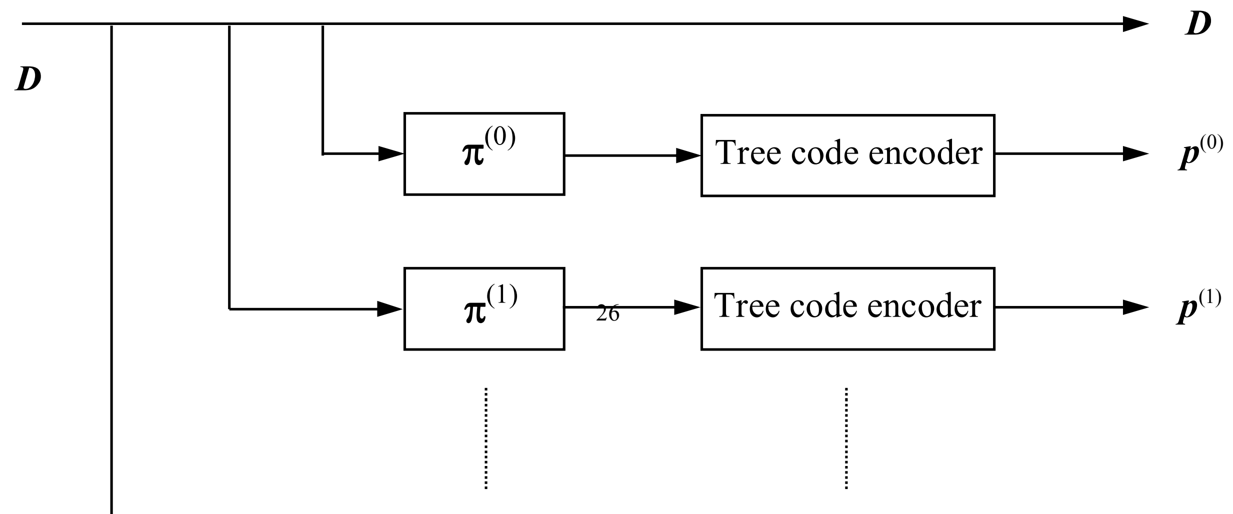
One of the important utilities for a bipartite graph is to define a linear error-correcting code, i.e., all the nodes of one class are associated with bits of the codeword; each of the nodes of the other is associated with a component code with the length as the degree of that node. The inherent merits of bipartite graphs are of particular useful for the belief propagation algorithm where the extrinsic information exchanges between neighbour nodes of different classes, i.e., message-passing decoder that will be introduced in the Chapter 5.

Definition 2.4 The code is called a tree code if the corresponding Tanner graph does not contain any loop. □



Actually, we have intuitively used this concept before in this chapter. If the individual tree codes (usually identical) are further concatenated by a proper manner, we call this sort of codes as concatenated tree (CT) codes [26][39]. An illustrative Tanner graph of a tree code is shown in Fig. 2.11, which is encoded from the bottom upward by the same way. Evidently, all the component codes are tree codes, two of which are depicted by the dashed blocks. In this sense, the illustrated tree code can be also viewed as a loop-free CT code.

Compared with the Tanner graphs of LDPC and CT codes shown by Fig. 2.10(a) and Fig. 2.11, respectively, we may observe that both of them surely exist some interesting similarities. Graphically, a CT code can be seen as a special LDPC code consisting of several trees with large span.



A practical CT code is designed by parallel concatenation of M constituent tree codes in a turbo-type fashion [26][39-40] as shown in Fig. 2.12, where the information and parity sequences are denoted by $\boldsymbol{D}$, and $\boldsymbol{p}^{(m)}$ (m=0, 1, 2, …, M−1), respectively. This resultant CT code can still be viewed as an LDPC code for small M. The degrees of the information nodes in the overall CT code are increased by M times from those in the constituent codes. Generically, CT and LDPC codes usually have loopy graphs, even for good LDPC codes [1][2][22-27].