Final solutions

Problem F.1 (50 points)

In this problem, we will investigate ML decoding using the standard VA on a minimal conventional trellis, or using a modified VA on a tail-biting trellis.

Consider the $(6, 3, 3)$ *binary linear code* $\mathcal{C}$ *that is generated by the following generator matrix:*

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

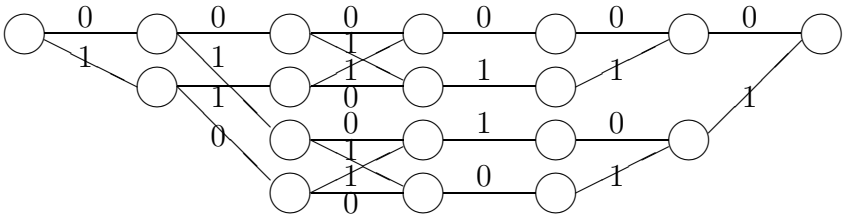
(a) Find the state and branch complexity profiles of an unsectionalized minimal trellis for $\mathcal{C}$*. Draw the corresponding minimal trellis for* $\mathcal{C}$*, and label each branch with the corresponding output symbol. Verify that there is a one-to-one map between the codewords of* $\mathcal{C}$ *and the paths through the trellis.*

We first find a trellis-oriented generator matrix for $\mathcal{C}$. Replacing the third generator by the sum of the first and third generator, we arrive at the following generator matrix:

$$G' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Since the starting times and ending times of all generators are distinct, G' is trellis-oriented.

Using the active intervals of the generators in G', we find that the state dimension profile of a minimal trellis for $\mathcal{C}$ is $\{0, 1, 2, 2, 2, 1, 0\}$, and the branch dimension profile is $\{1, 2, 3, 2, 2, 1\}$. Explicitly, a minimal trellis is



We verify that there are eight paths through the trellis corresponding to the codewords $\{000000, 111000, 001110, 110110, 011011, 100011, 010101, 101101\}$.

(b) Show that the following unsectionalized 2-state tail-biting trellis (TBT) realizes $\mathcal{C}$. (Recall that in a TBT there may be more than one state in $\Sigma_0 = \Sigma_6$, the starting and ending state space, and that the valid paths are those that start and end in the same state in $\Sigma_0 = \Sigma_6$.) Verify that there is a one-to-one map between the codewords of $\mathcal{C}$ and the valid paths through the tail-biting trellis.

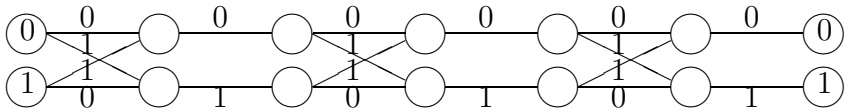
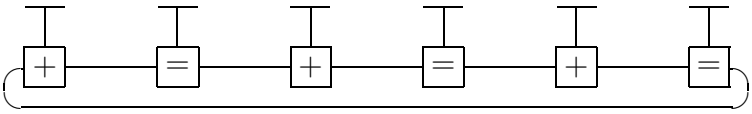


Figure 1. Tail-biting trellis for $\mathcal{C}$.

Here the two states at state time 6 should be regarded as the same as the two states at state time 0. The valid paths are those that start and end in state 0, corresponding to the four codewords $\{000000, 111000, 001110, 110110\}$, and those that start and end in state 1, corresponding to the four codewords $\{011011, 100011, 010101, 101101\}$. Thus we verify that this TBT realizes $\mathcal{C}$.

Draw a normal graph of this tail-biting trellis realization of $\mathcal{C}$.

The normal graph of a tail-biting trellis looks the same as that of a conventional trellis, except that the two ends are joined so that the ending state variable is the same as the starting state variable. In this case all states are binary, and the constraints are alternately single-parity-check and repetition constraints. So the normal graph looks like this:



Normal graph of a tail-biting trellis for $\mathcal{C}$.

Note that G is a TBT-oriented generator matrix for $\mathcal{C}$, in the following sense. If the active intervals of the generators in G are viewed on an end-around basis— *i.e.,* $[0, 2]$, $[2, 4]$, and $[4, 0]$— then there is only one generator active at each state time, so the corresponding state dimension profile is $\{1, 1, 1, 1, 1, 1\}$. Similarly, these end-around activity intervals imply a branch dimension profile of $\{2, 1, 2, 1, 2, 1\}$. The TBT above is generated by these three end-around generators in the same way as a conventional trellis is generated by conventional trellis-oriented generators.

(c) Propose a modification to the Viterbi algorithm that finds the maximum-likelihood (ML) codeword in $\mathcal{C}$, using the tail-biting trellis of part (b). Compare the complexity of your modified VA to that of the standard VA operating on the minimal conventional trellis of part (a).

We can run the standard Viterbi algorithm on the two subtrellises consisting of the subsets of valid paths that start and end in states 0 and 1, respectively. This will find the ML codeword among the two corresponding subsets of 4 codewords. We can then choose the best of these two survivors as the ML codeword.

Roughly, this requires running the VA twice on two two-state trellises, compared to one four-state trellis, so in terms of state complexity the complexity is roughly the same. The maximum branch complexity of the two-state trellises is 4, whereas that of the four-state trellis is 8, so again the branch complexity is roughly the same. Finally, if we do a detailed count of additions and comparisons, we find that both methods require 22 additions and 7 comparisons— exactly the same. We conclude that the simpler tail-biting trellis requires the same number of operations for ML decoding, if we use this modified VA.

(d) For a general (n, k) linear code $\mathcal{C}$ with a given coordinate ordering, is it possible to find a minimal unsectionalized tail-biting trellis that simultaneously minimizes the complexity of each of the n state spaces over all tail-biting trellises? Explain.

The example above shows that this is not possible. The conventional trellis for $\mathcal{C}$ is also a tail-biting trellis for $\mathcal{C}$ with a state space $\Sigma_0 = \Sigma_6 = \{0\}$ of size 1. Thus, as a TBT, the conventional trellis minimizes the state space size at state time 0, but at the cost of state spaces larger than 2 at other times. We have already noted that it is not possible to construct a TBT for $\mathcal{C}$ that achieves the state space dimension profile $\{0, 1, 1, 1, 1, 1\}$, which is what would be necessary to achieve the minima of these two profiles simultaneously at all state times.

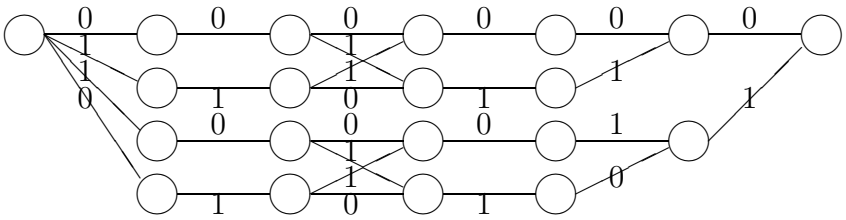
Moreover, we can similarly obtain a tail-biting trellis with a state space of size 1 at any state time, by cyclically shifting the generators in order to construct a minimal conventional trellis that starts and ends at that state time. But there is obviously no way to obtain a minimal state space size of 1 at all state times simultaneously.

(e) Given a general linear code $\mathcal{C}$ and an unsectionalized tail-biting trellis for $\mathcal{C}$, propose a modified VA that performs ML decoding of $\mathcal{C}$ using the tail-biting trellis. In view of the cut-set bound, is it possible to achieve a significant complexity reduction over the standard VA by using such a modified VA?

In general, we can run the standard VA on the $|\Sigma_0|$ subtrellises consisting of the subsets of valid paths that start and end in each state in Σ_0. This will find the ML codeword among each of the $|\Sigma_0|$ corresponding codeword subsets. We can then choose the best of these $|\Sigma_0|$ survivors as the ML codeword.

The complexity of this modified VA is of the order of $|\Sigma_0|$ times the complexity of VA decoding a subset trellis. The subset trellis state complexity is the state complexity $\max_k |\Sigma_k^{\mathcal{C}'}|$ for a conventional trellis for the code $\mathcal{C}'$ that is generated by the generators of the TBT-oriented generator matrix for $\mathcal{C}$ that do not span state time 0. Choose a cut set consisting of state time 0 and state time k for any other $k, 0 < k < n$. By the cut-set bound, the total number of generators that span state time either 0 or k is not less than the minimal total number of generators that span state time k in a conventional trellis for $\mathcal{C}$. Therefore $|\Sigma_0||\Sigma_k^{\mathcal{C}'}| \geq |\Sigma_k^{\mathcal{C}}|$ for any $k, 0 < k < n$. But this implies that $\max_k |\Sigma_0||\Sigma_k^{\mathcal{C}'}| \geq \max_k |\Sigma_k^{\mathcal{C}}|$. Thus no reduction in aggregate state complexity can be obtained by the modified VA.

An easier way of seeing this is to notice that the operation of the modified VA is the same as that of a standard VA on a nonminimal conventional trellis consisting of $|\Sigma_0|$ parallel subtrellises, joined only at the starting and ending nodes. For example, for part (c) we can think of a standard VA operating on the following nonminimal conventional trellis:



Since this is merely another conventional trellis, in general nonminimal, the modified VA operating on this trellis must clearly be at least as complex as the standard VA operating on the minimal conventional trellis.

We conclude that if we use the modified VA on a TBT to achieve ML decoding, then we cannot achieve any savings in decoding complexity. On the other hand, iterative decoding on the TBT may be less complex, but in general will not give ML decoding performance.

Problem F.2 (60 points)

In this problem, we will analyze the performance of iterative decoding of a rate-1/3 repeat-accumulate (RA) code on a binary erasure channel (BEC) with erasure probability p, in the limit as the code becomes very long ($n \to \infty$).

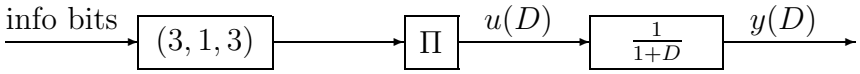


Figure 2. Rate-$\frac{1}{3}$ RA encoder.

The encoder for the rate-1/3 RA code is shown in Figure 2 above, and works as follows. A sequence of information bits is first encoded by an encoder for a $(3, 1, 3)$ *repetition code, which simply repeats each information bit three times. The resulting sequence is then permuted by a large pseudo-random permutation* Π*. The permuted sequence* $u(D)$ *is then encoded by a rate-*$1/1$ *2-state convolutional encoder with input/output relation* $y(D) = u(D)/(1 + D)$*;* i.e., *the input/output equation is* $y_k = u_k + y_{k-1}$*, so the output bit is simply the "accumulation" of all previous input bits (mod 2).*

(a) Show that this rate-$\frac{1}{3}$ RA code has the normal graph of Figure 3.

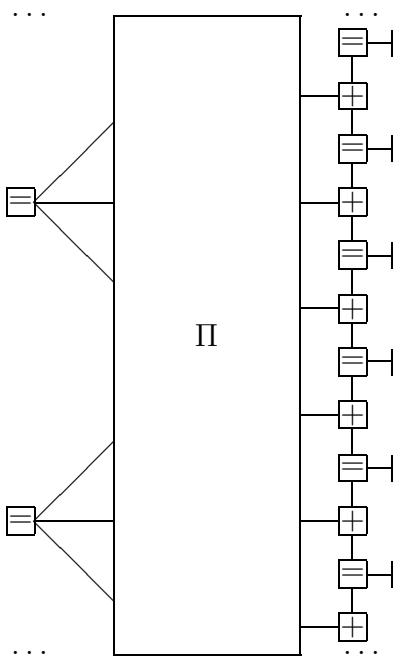


Figure 3. Normal graph of rate-$\frac{1}{3}$ RA code.

The left-side nodes of Figure 2 represent the repetition code. Since the original information bits are not transmitted, they are regarded as hidden state variables, repeated three times. The repeated bits are permuted in the permutation Π. On the right side, the permuted bits u_k are the input bits and the y_k are the output bits of a 2-state trellis, whose states are the output bits y_k. The trellis constraints are represented explicitly by zero-sum nodes that enforce the constraints $y_k + u_k + y_{k-1} = 0$.

(b) Suppose that the encoded bits are sent over a BEC with erasure probability p. Explain how iterative decoding works in this case, using a schedule that alternates between the left constraints and the right constraints.

The outputs of a BEC are either known with certainty or completely unknown (erased). The sum-product algorithm reduces to propagation of known variables through the code graph. If any variable incident on a repetition node becomes known, then all become known. On the other hand, for a zero-sum node, all but one incident variable must be known in order to determine the last incident variable; otherwise all unknown incident variables remain unknown.

In detail, we see that initially an input bit u_k becomes known if and only if the two adjoining received symbols, y_k and y_{k-1}, are unerased. After passage through Π, these known bits propagate through the repetition nodes to make all equal variables known. After passage through Π, the right-going known bits are propagated through the trellis, with additional input or output bits becoming known whenever two of the three bits in any set $\{y_{k-1}, u_k, y_k\}$ become known. Known input bits u_k are then propagated back through Π, and so forth.

(c) Show that, as $n \to \infty$, if the probability that a left-going iterative decoding message is erased is $q_{r\to\ell}$, then the probability that a right-going messages is erased after a left-side update is given by

$$q_{\ell\to r} = (q_{r\to\ell})^2.$$

As $n \to \infty$, for any fixed number m of iterations, we may assume that all variables in the m-level iteration tree are independent.

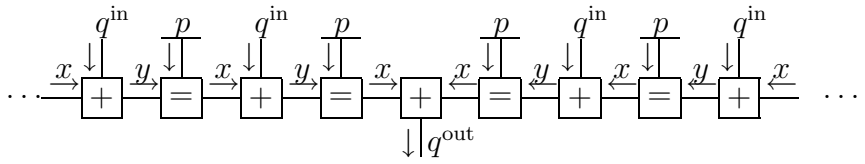
A right-going message is erased if and only if both left-going messages that are incident on the same repetition node is erased. Thus if these two variables are independent, each with probability $q_{r\to\ell}$ of erasure, then $q_{\ell\to r} = (q_{r\to\ell})^2$.

(d) Similarly, show that if the probability that a right-going iterative decoding message is erased is $q_{\ell\to r}$, then the probability that a left-going message is erased after a right-side update is given by

$$q_{r\to\ell} = 1 - \frac{(1-p)^2}{(1-p+pq_{\ell\to r})^2}.$$

[Hint: observe that as $n \to \infty$, the right-side message probability distributions become invariant to a shift of one time unit.]

For a particular input bit, the messages that contribute to the calculation of the outgoing message look like this:



Here p denotes the probability that an output bit y_k will be erased on the channel, q^{in} denotes the probability $q_{\ell\to r}$ that an input bit u_k will still be erased after the previous iteration, and q^{out} denotes the probability $q_{r\to\ell}$ that the bit that we are interested in will be erased after this iteration. Again, as $n \to \infty$, we may assume that all of these probabilities are independent.

Following the hint, we use the symmetry and time-invariance of the trellises on either side of u_k to assert that as $n \to \infty$ the probability of erasure x in all of the messages marked with x will be the same, and similarly that the probability of erasure y in all of the messages marked with y will be the same.

The relations between these probabilities are then evidently as follows:

$$x = py; \qquad 1 - y = (1 - q^{\text{in}})(1 - x); \qquad 1 - q^{\text{out}} = (1 - x)^2.$$

Solving the first two equations, we obtain

$$y = \frac{q^{\text{in}}}{1 - p + pq^{\text{in}}}, \qquad x = \frac{pq^{\text{in}}}{1 - p + pq^{\text{in}}},$$

and thus

$$q^{\text{out}} = 1 - \frac{(1-p)^2}{(1-p+pq^{\text{in}})^2}.$$

(e) Using a version of the area theorem that is appropriate for this scenario, show that iterative decoding cannot succeed if $p \geq \frac{2}{3}$.

The area under the curve of part (c) is

$$\int_0^1 q^2 dq = \frac{1}{3}.$$

The area under the curve of part (d) is

$$1 - \int_0^1 \frac{(1-p)^2}{(1-p+pq)^2} dq = 1 - \frac{(1-p)^2}{p} \left[\frac{1}{1-p+pq} \right]_0^1 = p.$$

Iterative decoding will succeed if and only if the two curves do not cross. In order for the two curves not to cross, the sum of these two areas must be less than the area of the EXIT chart; *i.e.,*

$$\frac{1}{3} + p < 1,$$

which is equivalent to $p < \frac{2}{3}$; *i.e.,* the capacity $1-p$ of the BEC, namely $1-p$, must be greater than the rate of the RA code, namely $\frac{1}{3}$.

(For example, in Figure 4 below, the area above the top curve is $\frac{1}{3}$, whereas the area below the bottom curve is $\frac{1}{2}$.)

(f) The two curves given in parts (c) and (d) are plotted in the EXIT chart below for $p = 0.5$. *Show that iterative decoding succeeds in this case.*

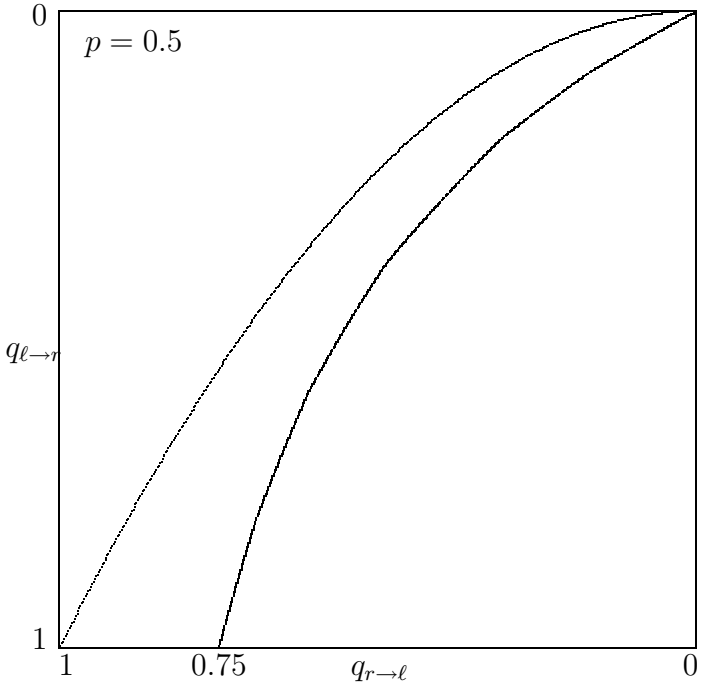


Figure 3. EXIT chart for iterative decoding of a rate-$\frac{1}{3}$ RA code on a BEC with $p = 0.5$.

The two curves do not cross, so iterative decoding starting at $(q_{r\to\ell}, q_{\ell\to r}) = (1, 1)$ must succeed (reach $(q_{r\to\ell}, q_{\ell\to r}) = (0, 0)$).

(g) [Optional; extra credit.] Determine whether or not iterative decoding succeeds for $p = 0.6$.

The easiest way to determine whether iterative decoding succeeds for $p = 0.6$ is to simulate it using the equations above. We obtain

$q_{\ell \to r}$	$q_{r \to \ell}$
1.0	0.84
0.706	0.764
0.584	0.716
0.512	0.680
0.463	0.652
0.425	0.627
0.393	0.604
0.365	0.582
0.339	0.561
0.314	0.538
0.290	0.514
0.264	0.487
0.237	0.456
0.208	0.419
0.176	0.373
0.139	0.316
0.100	0.244
0.059	0.157
0.025	0.070
0.005	0.015
0.000	0.001
...	...

Thus iterative decoding succeeds fairly easily for $p = 0.6$, even though the capacity of a BEC with $p = 0.6$ is only 0.4, not much greater than the rate of the RA code.

It is possible therefore that irregular RA codes may be capacity-approaching for the BEC. Even with regular codes, the two EXIT curves are already quite well matched. However, note that only the left degrees can be made irregular, which limits design flexibility.

Problem F.3 (40 points)

For each of the propositions below, state whether the proposition is true or false, and give a proof of not more than a few sentences, or a counterexample. No credit will be given for a correct answer without an adequate explanation.

(a) The Euclidean image of an (n, k, d) binary linear block code is an orthogonal signal set if and only if $k = \log_2 n$ and $d = n/2$.

FALSE. The Euclidean images of two binary words are orthogonal if and only if $d = n/2$, so all codewords must be at distance $d = n/2$ from each other. In a linear code, this happens if and only if all nonzero codewords have weight $d = n/2$. However, all that is specified is that the minimum distance is $d = n/2$, which is necessary but not sufficient. The smallest counterexample is a $(4, 2, 2)$ code; *e.g.,* $\{0000, 1100, 0011, 1111\}$.

(b) If a possibly catastrophic binary rate-$1/n$ linear convolutional code with polynomial encoder $\mathbf{g}(D)$ is terminated to a block code $\mathcal{C}_\mu = \{u(D)\mathbf{g}(D) \mid \deg u(D) < \mu\}$, then a set of μ shifts $\{D^k\mathbf{g}(D) \mid 0 \le k < \mu\}$ of $\mathbf{g}(D)$ is a trellis-oriented generator matrix for $\mathcal{C}_\mu$.

TRUE. The μ shifts $G = \{D^k\mathbf{g}(D) \mid 0 \le k < \mu\}$ form a set of generators for $\mathcal{C}_\mu$, since every codeword may be written as $\sum_0^{\mu-1} u_k(D^k\mathbf{g}(D))$ for some binary μ-tuple $(u_0, u_1, \ldots, u_{\mu-1})$. If the starting and ending times of $\mathbf{g}(D)$ occur during n-tuple times del $\mathbf{g}(D)$ and $\deg \mathbf{g}(D)$, respectively, then the starting time of $D^k\mathbf{g}(D)$ occurs during n-tuple time del $\mathbf{g}(D) + k$, and its ending time occurs during n-tuple time $k + \deg \mathbf{g}(D)$. Thus the starting times of all generators are distinct, and so are all ending times, so the set G is trellis-oriented.

(c) Suppose that a codeword $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ in some code $\mathcal{C}$ is sent over a memoryless channel, such that the probability of an output $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ is given by $p(\mathbf{r} \mid \mathbf{y}) = \prod_1^n p(r_i \mid y_i)$. Then the a posteriori *probability $p(Y_1 = y_1, Y_2 = y_2 \mid \mathbf{r})$ is given by*

$$p(Y_1 = y_1, Y_2 = y_2 \mid \mathbf{r}) \propto p(Y_1 = y_1 \mid r_1)p(Y_2 = y_2 \mid r_2)p(Y_1 = y_1, Y_2 = y_2 \mid r_3, \ldots, r_n).$$

TRUE. This may be viewed as an instance of Equation (12.6) of the course notes, where we take (Y_1, Y_2) as a single symbol variable. Alternatively, this equation may be derived from first principles. We have

$$p(Y_1 = y_1, Y_2 = y_2 \mid \mathbf{r}) = \sum_{\mathbf{y} \in \mathcal{C}(y_1, y_2)} p(\mathbf{y} \mid \mathbf{r}) \propto \sum_{\mathbf{y} \in \mathcal{C}(y_1, y_2)} p(\mathbf{r} \mid \mathbf{y}) = \sum_{\mathbf{y} \in \mathcal{C}(y_1, y_2)} \prod_{i \in \mathcal{I}} p(r_i \mid y_i),$$

where $\mathcal{C}(y_1, y_2)$ denotes the subset of codewords with $Y_1 = y_1, Y_2 = y_2$. We may factor $p(r_1 \mid y_1)p(r_2 \mid y_2)$ out of every term on the right, yielding

$$p(Y_i = y_i \mid \mathbf{r}) \propto p(r_1 \mid y_1)p(r_2 \mid y_2) \left(\sum_{\mathbf{y} \in \mathcal{C}(y_1, y_2)} \prod_{i=3}^{n} p(r_i \mid y_i) \right).$$

But now by Bayes' rule $p(Y_1 = y_1 \mid r_1) \propto p(r_1 \mid y_1)$, $p(Y_2 = y_2 \mid r_2) \propto p(r_2 \mid y_2)$ (assuming equiprobable symbols), and we recognize that the last term is proportional to the *a posteriori* probability $p(Y_1 = y_1, Y_2 = y_2 \mid r_3, \ldots, r_n)$.

(d) Let $\mathcal{C}$ be the binary block code whose normal graph is shown in the figure below. All left constraints are repetition constraints and have the same degree d_λ (not counting the dongle); all right constraints have the same degree d_ρ and are given by a binary linear $(d_\rho, \kappa d_\rho)$ constraint code $\mathcal{C}_c$. Assuming that all constraints are independent, the rate of $\mathcal{C}$ is

$$R = 1 - d_\lambda(1-\kappa).$$

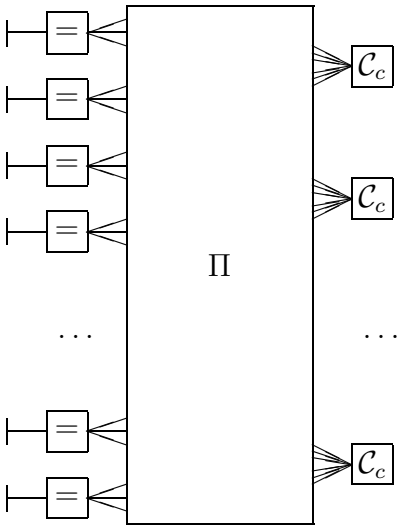


Figure 5. Normal graph realization of $\mathcal{C}$, with left repetition constraints of degree d_λ and right constraint codes $\mathcal{C}_c$ of degree d_ρ.

TRUE. First, if the length of $\mathcal{C}$ is n, then there are n left constraints and nd_λ left edges. If there are m right constraints, then there are md_ρ right edges, which must equal the number of left edges. Thus $m = nd_\lambda/d_\rho$.

Second, each constraint code $\mathcal{C}_c$ consists of the set of all binary d_ρ-tuples that satisfy a set of $(1-\kappa)d_\rho$ parity-check equations. Thus $\mathcal{C}$ is equivalent to a regular LDPC code of length n with $m(1-\kappa)d_\rho = nd_\lambda(1-\kappa)$ parity checks. Assuming that all checks are independent, the rate of such a code is

$$R = 1 - \frac{nd_\lambda(1-\kappa)}{n} = 1 - d_\lambda(1-\kappa).$$