Problem 4.14

Suppose that the K users can be divided in different S groups such that within each group users are synchronized but users in different groups are not mutually synchronized. Find a maximum-likelihood multiuser detector with time complexity per bit equal to $O(S2^K/K)$

Proposed solution

The multiuser detector described next is a modified version of the jointly optimum detector for asynchronous users. We (section 4.2) essentially take profit that the $\mathbf{H} = \mathbf{b}^T\mathbf{R}\mathbf{b}$ matrix for K asynchronous users is even more sparse given the setting of this problem, and vary the sizes of the state vectors at different stages of the trellis.

Assume each of the S groups of synchronized users has L_i elements, where $i = 1..S$; the indices of users in (the mutually synchronous) group i belong to the set $G_i = \{j \text{ integer } : (1 + \sum_{j=1}^{i-1} L_j) \leq j \leq (\sum_{j=1}^{i} L_j)\}$. To ease notation, we'll define the function g as: $g(j) = i$ if $j \in G_i$ i.e. g(j) returns the index of the group to which j belongs.

First, we observe the structure of $\mathbf{R[1]}$;

$$\text{for } i < j, \rho_{ij} = 0 \text{ if } i, j \in G_n \text{ for some } n;$$

[apply formula 2.13b to synchronous signals]

From this and 2.109 $R_{jk}[1] = 0$ if $g(k) \leq g(j)$. To ease understanding here is $\mathbf{R[1]}$ for $G_1 = \{1,2,3\}, G_2 = \{4,5,6\}, G_3 = \{7,8\}, S = 3$:

$$\begin{pmatrix} 0 & 0 & 0 & \rho_{41} & \rho_{51} & \rho_{61} & \rho_{71} & \rho_{81} \\ 0 & 0 & 0 & \rho_{42} & \rho_{52} & \rho_{62} & \rho_{72} & \rho_{82} \\ 0 & 0 & 0 & \rho_{43} & \rho_{53} & \rho_{63} & \rho_{73} & \rho_{83} \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho_{74} & \rho_{84} \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho_{75} & \rho_{85} \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho_{76} & \rho_{86} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We define the function $r(i) = \sum_{j=1}^{g(i)} L_j + 1 = min\{m, m \in G_{g(i)}\} + L_{g(i)} + 1$. Then we can claim that $R_{jk}[1] = 0$ for all $k < r(j)$. For example, in the above matrix:

$r(1) = 4,\;$ and $R_{1k}[1] = 0$ if $k < 4$
$r(2) = 4,\;$ and $R_{2k}[1] = 0$ if $k < 4$
$r(3) = 4,\;$ and $R_{3k}[1] = 0$ if $k < 4$
$r(4) = 7,\;$ and $R_{4k}[1] = 0$ if $k < 7$
$r(5) = 7,\;$ and $R_{5k}[1] = 0$ if $k < 7$
$\ldots$

With this observation [and noticing that $\mathbf{H}$ is a symmetric matrix], we can modify formula 4.33 to read:

$$\mathbf{b}^T\mathbf{H}\mathbf{b} = \sum_{j=1-MK}^{MK+K} b_j \left[A^2_{\kappa(j)} b_j + 2 \sum_{l=j-K+1}^{j-1} b_l h_{j,l} \right] \tag{1}$$

$$= \sum_{j=1-MK}^{MK+K} A_{\kappa(j)} b_j \left[A_{\kappa(j)} b_j + 2 \sum_{l=j-K+1}^{j-\kappa(j)} b_l h_{jl} + 2 \sum_{l=j-\kappa(j)+1}^{j-1} b_l h_{jl} \right] \tag{2}$$

$$= \sum_{j=1-MK}^{MK+K} A_{\kappa(j)} b_j \left[A_{\kappa(j)} b_j + 2 \sum_{l=j-K+1}^{j-\kappa(j)} b_l A_{\kappa(l)} \rho_{\kappa(l)\kappa(j)} + 2 \sum_{l=j-\kappa(j)+1}^{j-1} b_l h_{jl} \right] \tag{3}$$

$$= \sum_{j=1-MK}^{MK+K} A_{\kappa(j)} b_j \left[A_{\kappa(j)} b_j + 2 \sum_{l=\kappa(j)+1}^{K} b_{l-j+K+\kappa(j)} A_l \rho_{l\kappa(j)} + 2 \sum_{l=j-\kappa(j)+1}^{j-1} b_l h_{jl} \right] \tag{4}$$

$$= \sum_{j=1-MK}^{MK+K} A_{\kappa(j)} b_j \left[A_{\kappa(j)} b_j + 2 \sum_{l=r(\kappa(j))}^{K} b_{l-j+K+\kappa(j)} A_l \rho_{l\kappa(j)} + 2 \sum_{l=j-\kappa(j)+1}^{j-1} b_l h_{jl} \right] \tag{5}$$

$$= \sum_{j=1-MK}^{MK+K} b_j \left[A^2_{\kappa(j)} b_j + 2 \sum_{l=j-K+r(\kappa(j))-\kappa(j)}^{j-1} b_l h_{j,l} \right] \tag{6}$$

where eq. 1 is a modified version of 4.33, eq. 3 splits $\mathbf{H}$ entries between locations dependent on $\mathbf{R[1]}$ and those dependent on $\mathbf{R[0]}$. Eq. 4 is obtained through a change of variable, and observing that $\kappa(l - j + K + \kappa(j)) = l$ since

$1 < l \leq K$. Eq. 5 makes use of the observation on the zero entries of $\mathbf{R[1]}$. Finally, eq. 6 undoes the effect of the change of variables.

The structure of eq. 6 implies that - in the trellis - we don't need vector states of size K-1, but lower (varying with j) sizes may suffice.

Let us examine the sizes of the state vectors needed as j goes through values $j_0, j_0+1, \ldots, j_0+K-1$, with $\kappa(j_0) = 1$. To retain sufficient information between stages of the trellis, we shall store all the b_j-s that appear in eq. 6 (this may be a bit wasteful in terms of state vector bits, but makes the problem more easily tractable). Finally, at stage j we need $t(j) = (j-1)-(j-K+r(\kappa(j))-\kappa(j))+1 = K - r(\kappa(j)) + \kappa(j))$ bits.

We define one last function that takes as argument a user index and returns the index of the user within its own group. Let $q(j) = j - \sum_{i=1}^{g(j)-1} +1$. Thus, in the example we had on the first page, $q(1) = 1, q(2) = 2, q(3) = 3, q(4) = 1, q(5) = 2, \ldots$ We recogize that $r(\kappa(j)) - \kappa(j)) = L_{g(\kappa(j))} - q(j) + 1$ hence $t(j) = (K-1) - L_{g(\kappa(j))} + q(j)$

As shown in the textbook on page 172, an M-dimensional state requires $C2^M$ operations. To justify that this remains true even when state sizes vary, note that (using a forward dynamic implementation) vector sizes may increase by 1 (within a group of synchronized users), or decrease (by 1 or more at transitions between groups); for that reason, *each* state at stage j would be 'tapped' exactly twice in the attempt to generate $J_{j+1}(\mathbf{x}_{j+1})$ for stage $j+1$ (see p. 172 in the textbook)[1]. Summing over values of j from j_0 to $j_0 + K - 1$ – or, equivalently, from 1 to K because $t(j)$ depends only on $\kappa(j)$ – we get the number of operations:

$$C \sum_{j=1}^{K} 2^{t(j)} = C \sum_{i=1}^{S} \sum_{l=1}^{L_i} 2^{K-1-L_i+l} = C2^{K-1} \sum_{i=1}^{S} \sum_{l=1}^{L_i} 2^{-(L_i-l)}$$

$$= C2^{K-1} \sum_{i=1}^{S} \sum_{l=0}^{L_i-1} 2^{-l} = C2^{K} \sum_{i=1}^{S} (1 - 2^{-L_i}) \quad (7)$$

The term $1-2^{-L_i}$ is bounded between 0.5 and 1, so the processing complexity for computing K [periodically repeated] stages of the trellis is $O(S2^K)$. Each stage corresponds to a bit that is eventually decoded, so we get a complexity per bit of $O(S2^K/K)$

[1] strictly speaking we need to take into account the increased complexity in the computation of *max* over possibly more than two arguments, but with a proper implementation – a tree – there will be an increase of a factor of at most 2, and this does not affect the final complexity computation