IMPLANTATION D'UNE COUCHE PHYSIQUE WIFI
Sciences du Numérique
Deuxième année - Parcours Systèmes de télécom

1 Introduction

Le standard IEEE 802.11 normalise les réseaux locaux sans fil. Les versions IEEE 802.11a, IEEE 802.11g et suivantes (n, ac, ax) sont basées sur la modulation OFDM afin de pouvoir augmenter les débits, par rapport aux versions précédentes (IEEE802.11 ou legacy et IEEE 802.11b) sur des canaux sélectifs en fréquence.

Les versions a, g sont les premières utilisant l'OFDM pour passer des débits pouvant aller jusqu'à 54 Mbps.

Leur différence essentielle se situe au niveau des bandes utilisées pour la transmission : Unlicensed National Information Infrastructure (UNII) autour de 5 GHz pour le a et Industrielle Scientifique et Médicale (ISM) autour de $2,4$ GHz pour le g. Le travail à 5 Ghz permet d'accèder à davantage d'espace fréquentiel : jusqu'à 12 canaux (aux Etats Unis, 8 en Europe) sans recouvrement d'une largeur de 25 MHz, largeur de bande adoptée pour parvenir à des débits allant jusqu'à 54 Mbps (20 MHz de bande utile + 5 MHz d'extension de bande due au préfixe cyclique). La bande ISM à 2.4 GHz ne permet de définir que 3 canaux sans recouvrement d'une largeur de 25 MHz.

Dans les deux cas (a et g) un code convolutif est utilisé. Dans les versions suivantes apparaissent le codage LDPC, ainsi que les technologies MIMO, plus de flexibilité dans les paramètres OFDM, la possibilité d'agréger des canaux et de réaliser de la formation de faisceaux.

Nous vous demanderons, dans ce projet d'implanter la couche physique permettant de transporter les données lors d'une transmission WiFi au format IEEE802.11a.

Cela sera l'occasion d'étudier l'apport du codage canal : codage convolutif dans un premier temps, en comparant avec une chaine de transmission non codée, puis l'impact d'un codage canal plus performant tel que le codage LDPC utilisé dans les versions suivantes des couches physiques du WiFi.

2 Couche physique IEEE 802.11a

Le standard IEEE802.11a ([1]) est un standard qui définit la couche physique du WiFi travaillant dans les bandes UNII autour de 5 GHz. Cette couche physique est composée de deux sous couches (figure 1) : la sous couche PLCP (Physical Layer Convergence Procedure) et la sous couche PMD (Physical Medium Dependant).

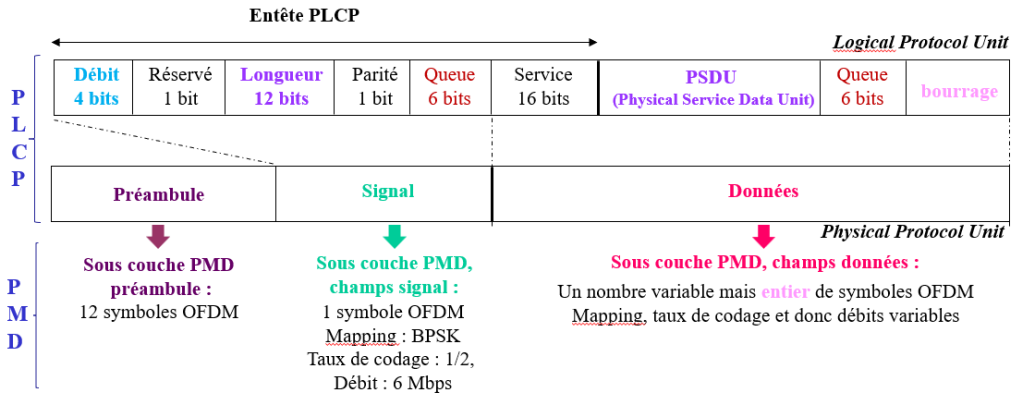


Figure 1: Sous couches de la couche physique du standard IEEE 802.11a

La sous couche PLCP est le ciment entre les trames MAC et la transmission radio hertzienne. Elle va ajouter aux données issues de la couche MAC (PSDU) des paramètres spécifiques à la couche physique.

La sous couche PMD est responsable de la transmission de chaque bit reçu de la sous couche PLCP par transmission hertzienne en utilisant une antenne. Elle va définir les caractéristiques et les méthodes d'émission des données à travers un support sans fil entre deux stations utilisant l'OFDM.

Une couche physique différente est définie au niveau PMD pour chacun des trois champs définis par la sous couche PLCP : préambule, signal et données. Nous nous intéresserons, dans ce projet, à celle associée au champ "données" qui est présentée dans la figure 2.

Figure 2: Couche physique du champs données du IEEE 802.11a

2.1 Embrouilleur

L'embrouillage a pour objectif d'uniformiser l'énergie dans le domaine fréquentiel. Il est réalisé grâce à un registre à décalage (voir figure 3) initialisé avec une séquence PN qu'il est possible de retrouver en réception grâce au champ service.

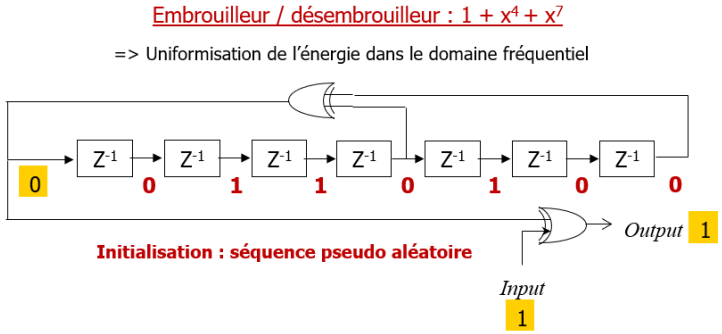


Figure 3: Couche PHY champ "Données" : embrouilleur

2.2 Codage canal

Le codeur utilisé est un codeur convolutif $(7, 1/2)$ de polynômes générateurs 133 et 171 (en octal). Grâce au poinçonnage il est possible d'obtenir des rendements de 2/3 et 3/4.
La figure 4 présente le codeur utilisé, tandis que la figure 5 présente les schémas de poinçonnage.

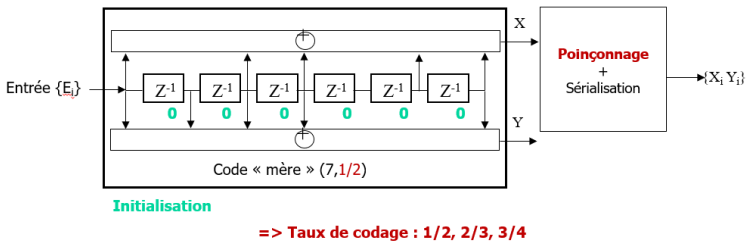


Figure 4: Couche PHY champ "Données" : codeur canal

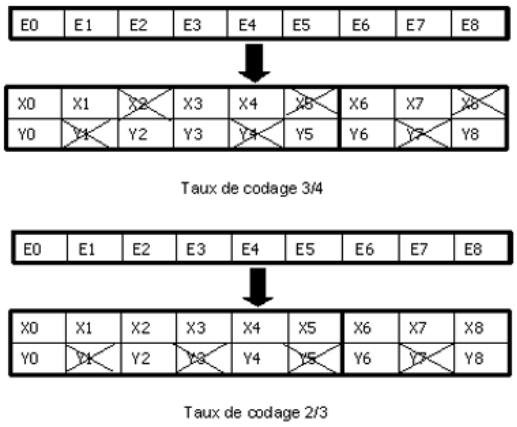


Figure 5: Couche PHY champ "Données" : schémas de poinçonnage

2.3 Entrelacement

L'entrelacement utilisé est un double entrelacement de type bloc.
La figure 6 présente un exemple d'entrelacement dans le cas d'une modulation 16-QAM envoyée sur les 48 porteuses utiles que nous avons dans le standard.

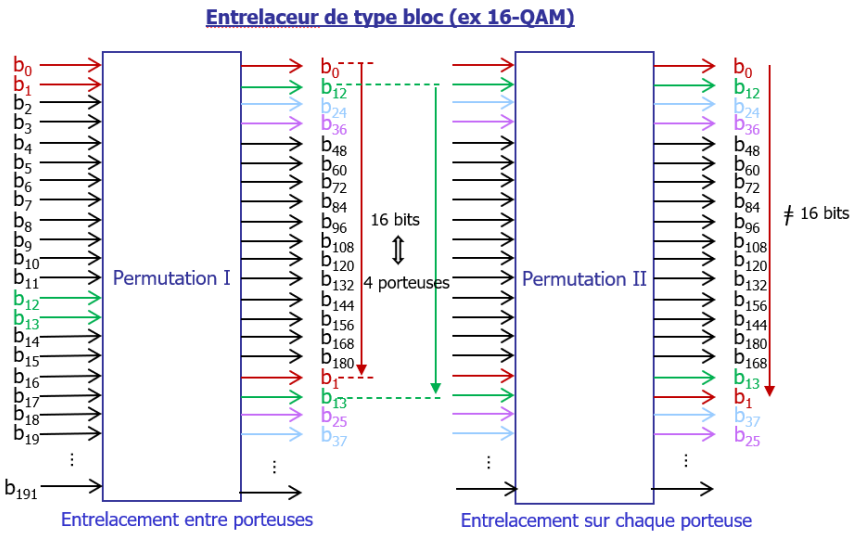


Figure 6: Couche PHY champ "Données" : entrelacement

4 bits sont envoyés par symbole en 16-QAM, ce qui donne $4 \times 48 = 192$ bits transportés sur les 48 porteuses utiles. On envoie un bloc de 192 bits au premier entrelaceur qui fait en sorte que deux bits consécutifs d'entrée soient espacés de 16 bits en sortie, soit 4 porteuses, en utilisant la règle suivante :

$$k_s = \left(\frac{N_{cbps}}{16}\right)(k_e \bmod 16) + floor\left(\frac{k_e}{16}\right),\ k_e = 0, 1, ..., N_{cbps} - 1$$

qui donne les indices des positions de sortie des bits, k_s, en fonction des indices des positions d'entrée des bits, k_e. N_{cbps} est le nombre de bits codés par symbole.
Nous avons ainsi en sortie, sur deux porteuses consécutives, des bits qui étaient séparés de 12 bits en entrée, soit une profondeur d'entrelacement de 12.
Le deuxième entrelaceur va entrelacer les 4 bits portés sur chaque porteuse, de manière à ce que deux bits consécutifs ne soient pas toujours envoyés sur la même position dans le symbole issu du mapping. Les indices des positions de sortie des bits, k_s,

sont donnés en fonction des indices des positions d'entrée des bits, k_e, de la manière suivante :

$$k_s = s floor\left(\frac{k_e}{s}\right) + \left(k_e + N_{cbps} - floor\left(\frac{16k_e}{N_{cbps}}\right)\right) mod\ s$$
$$k_e = 0, 1, ..., N_{cbps} - 1,\ s = max\left(\frac{N_{bpsc}}{2}, 1\right)$$

où N_{bpsc} est le nombre de bits par sous porteuse.

2.4 Mapping

Les mapping proposés dans le 802.11a sont BPSK, QPSK, 16-QAM et 64-QAM, toujours avec un mapping de Gray et une normalisation des constellations à une puissance de 1.

2.5 OFDM

La figure 7 présente la partie OFDM du standard.

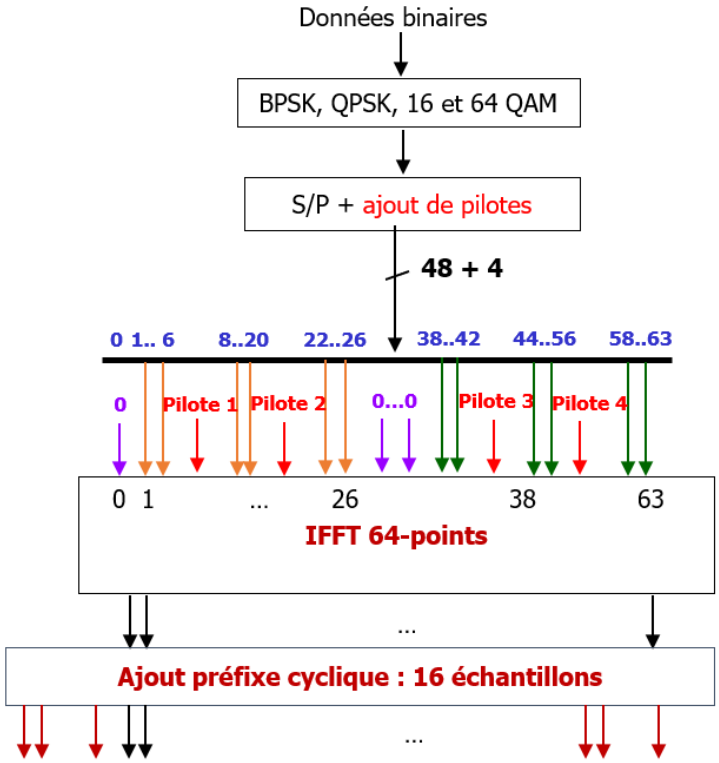


Figure 7: Couche PHY champ "Données" : OFDM

La bande allouée à une transmission WiFi au format IEEE 802.11a, B, est de 20 MHz (compromis entre les débits possibles et le nombre de canaux disjoints dans la bande UNII). L'étalement maximal des retards sur le trajet, τ_{max}, peut aller jusqu'à 200 ns, selon le contexte d'utilisation (bureau, appartement, entreprise, entrepôt...). La longueur du prefixe cyclique, T_{PC}, a été fixée à 800 ns (surdimensonnement pour faciliter la synchronisation) et la perte de débit utile, du fait de l'utilisation du prefixe cyclique, doit rester limitée à 20%. Nous avons donc :

$$\frac{1}{T'_s + T_{PC}} = 80\% \frac{1}{T'_s} \Rightarrow T'_s = \frac{0,8}{0,2} T_{PC} = 3200\text{ns}$$

où $R'_s = \frac{1}{T'_s}$ représente le débit des symboles OFDM, $T'_s = NT_s$ étant leur durée, avec T_s la période d'échantillonnage du signal OFDM.
L'espacement entre sous porteuses est de $\Delta f = \frac{1}{T'_s} = 0,3125$ MHz, ce qui donne un nombre de porteuse $N = \frac{B}{\Delta f} = 64$.

Parmi ces 64 porteuses :

- 48 vont transporter des données utiles,
- 4 seront des porteuses pilotes permettant d'estimer le canal de propagation. Elles ont des positions fixées données par la figure 7. Elles transportent des données issues d'un registre à décalage identique à celui utilisé pour l'embrouillage, mais avec une séquence d'initialisation différente (que des 1). Chaque sortie du registre est utilisé pour générer les données à envoyer sur les 4 pilotes : $\{1\ 1\ 1\ -1\}$ pour la sortie 0, $\{-1\ -1\ -1\ 1\}$ pour la sortie 1.
- 12 porteuses sont mises à 0 : porteuse à la fréquence 0 et porteuses sur les bords pour s'éloigner de la limite de Shannon et avoir une décroissance de puissance en bord de bande.

Le préfixe cyclique est constitué de $0,2 \times 64 = 16$ porteuses et génère donc une extension de bande de $16 \times 0,3125 = 5$ MHz.

Afin de diminuer la puissance hors bande, la norme IEEE802.11 propose, de manière optionnelle, un fenêtrage des symboles avec une fenêtre, $w_T(t)$, de type cosinus surélevé (voir figure 8) :

$$w_T(t) = \begin{cases} \sin^2\left(\frac{\pi}{2}\left(0,5 + \frac{t}{T_{TR}}\right)\right) & \text{si } -\frac{T_{TR}}{2} \leq t \leq \frac{T_{TR}}{2} \\ 1 & \text{si } \frac{T_{TR}}{2} \leq t < T - \frac{T_{TR}}{2} \\ \sin^2\left(\frac{\pi}{2}\left(0,5 - \frac{t-T}{T_{TR}}\right)\right) & \text{si } T - \frac{T_{TR}}{2} \leq t \leq T + \frac{T_{TR}}{2} \end{cases}$$

où $T = T'_s + T_{PC} = 4000$ ns.

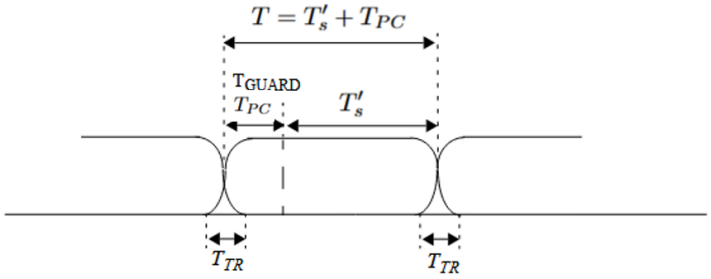


Figure 8: Couche PHY champ "Données" : fenêtrage (optionnel)

En numérique, avec $F_e = \frac{1}{T_s} = 20$ MHz et $T_{TR} = 100$ ns (2 échantillons) on a :

$$w_T(n) = w_T(nT_s) \begin{cases} 1 \text{ pour } 1 \leq n \leq 79 \\ 0,5 \text{ pour } n = 0 \text{ et } n = 80 \\ 0 \text{ ailleurs} \end{cases}$$

3 Implantation de la partie OFDM sur canal AWGN

Vous implanterez, dans un premier temps, une transmission utilisant l'OFDM, telle qu'elle est définie dans la norme IEEE 802.11a, en considérant un canal à bruit additif blanc et gaussien (AWGN).

On prendra donc :

- $N = 64$ porteuses dont $N_u = 48$ utiles avec des positions fixées par la norme, les autres seront, pour l'instant, toutes mises à 0, y compris les porteuses pilotes.
- un prefixe cyclique composé de 16 porteuses, même s'il n'est pas utile dans ce contexte.

Vous travaillerez avec la chaine passe-bas équivalente, en considérant un mapping QPSK sur chaque porteuse et avec une fréquence d'échantillonnage égale à R_s. Les symboles OFDM seront transmis à un débit $R'_s = \frac{R_s}{N}$, si N représente le nombre de porteuses utilisé.

En considérant que vous utilisez N_u porteuses pour transporter les données utiles et que vous mettez les autres à 0, vous pouvez introduire le bruit dans votre simulation avec la puissance suivante pour l'enveloppe complexe associée, $n_e(t)$:

$$P_{n_e} = \frac{N}{N_u} \times \frac{P}{2\log_2(M)\frac{E_b}{N_0}}$$

où P représente la puissance de l'enveloppe complexe associée au signal OFDM émis, M l'ordre de la modulation sur chaque porteuse et $\frac{E_b}{N_0}$ le rapport signal à bruit par bit à l'entrée du récepteur auquel on souhaite travailler.
Vérifiez que la courbe du taux d'erreur binaire (TEB) obtenu, en fonction de $\frac{E_b}{N_0}$ en dB, colle bien à celle du TEB théorique de la QPSK.

4 Implantation de la partie OFDM sur canal WiFi

Le canal WiFi est un canal multi trajets du fait des obstacles présents entre l'émetteur et le récepteur (retards, réflexions, diffractions...). **Nous allons considérer un environnement de bureau, pour lequel l'étalement maximal des retards est de l'ordre de** 200 **ns, que nous modéliserons en utilisant le modèle "TGn Model B" du groupe de travail qui a développé la norme WiFi 802.11n (MIMO-OFDM)**. Ce groupe a, en effet, défini une famille de modèles de canaux multitrajets réalistes destinés à permettre la simulation et la comparaison de systèmes MIMO-OFDM et à devenir un référentiel commun pour la recherche et l'industrie. Ces modèles sont appelés TGn Channel Models. Ils vont du modèle A au modèle F et décrivent des environnements de plus en plus riches en multi-trajets et de plus en plus sélectifs en fréquence.

Le TGn Model B définit un canal de Rayleigh sélectif en fréquence composé de $T = 9$ trajets avec les paramètres suivants :

- Etalements par rapport au premier trajet (pris comme référence), donnés en ns : [0 10 20 30 50 80 110 140 170]
- Puissances relatives sur les différents trajets, données en dB : $[0 - 5.4 - 10.8 - 16.2 - 21.6 - 27 - 32.4 - 37.8 - 43.2]$

En le supposant stationnaire, l'enveloppe complexe du signal signal reçu est donnée par :

$$y_e(t) = \sum_{k=0}^{T-1} \beta_k x_e(t - \tau_k) + n_e(t)$$

où β_k est un coefficient complexe représentant la combinaison des M trajets non séparables (avec un étalement des retards petit devant la durée T_s) formant chaque trajet séparable avec un délai moyen de τ_k ($|\tau_m - \tau_k| << T_s \ \forall m$) :

$$\beta_k = \sum_{m=0}^{M-1} \alpha_m e^{-j2\pi f_p \tau_m}$$

où f_p représente la fréquence porteuse. Afin d'obtenir un canal de Rayleigh β_k doit suivre une loi gaussienne à moyenne nulle.

Vous introduirez, dans votre chaine de transmission précédente, un canal WiFi multi trajets en utilisant une modélisation TGn Model B et en échantillonnant la réponse impulsionnelle à T_s puis vous tracerez la courbe du taux d'erreur binaire (TEB) obtenu, en fonction de $\frac{E_b}{N_0}$ en dB, en utilisant un égaliseur ZFE et/ou MMSE et en supposant le canal parfaitement connu.

5 Ajout du code convolutif

L'objectif de cette partie va être d'étudier l'apport du codage convolutif ajouté à la chaine précédente, en comparant les TEB obtenus avec et sans codage canal, avec codage canal en utilisant les deux types de décodage possibles : hard et soft et enfin en utilisant différents rendements grâce au poinçonnage.

Vous pouvez utiliser les fonctions Matlab : *poly2trellis.m* afin de génerer le treillis associé au code convolutif souhaité, *convenc.m* afin de réaliser le codage et *vitdec.m* pour réaliser un décodage de Viterbi. Il est possible de choisir le mode de décodage dans *vitdec.m*, de même qu'il est possible de fournir une matrice de poinçonnage aux fonctions Matlab *convenc.m* et *vitdec.m*.

Le standard IEEE802.11a définit un codage convolutif $(7, 1/2)$, de polynômes générateurs 133 et 171 en octal. Cependant, afin de vous familiariser avec le codage convolutif et les fonctions Matlab à utiliser, nous vous proposons, dans un premier temps, d'implanter un code plus simple, de type $(3, 1/2)$, avec des polynômes générateurs 5 et 7 en octal.

5.1 Exemple de codage convolutif $(3, 1/2)$

Le codeur convolutif $(3, 1/2)$, avec des polynômes générateurs 5 et 7 en octal, est donné par la figure 9.

Le treillis associé à ce code est donné par la figure 10. Il est construit grâce à la fonction *poly2trellis.m* de Matlab (voir figure 11). Le nombre d'entrées, b_n, possible est de 2 (0 et 1), le nombre de sorties, $s_{n1}s_{n2}$, possible est de 4 (00, 01, 10 et 11), le nombre d'états, $b_{n-1}b_{n-2}$, est de 4 et ils sont numérotés en octal de la manière suivante : état 0 = 00, état 1 = 01, état 2 = 10, état 3 = 11. *poly2trellis.m* fournit la table des états suivants et la table des sorties : figure 12. Une fois le treillis construit, vous pouvez le donner en paramètre à la fonction *convenc.m* de Matlab afin de réaliser le codage : *bitscodes=convenc(bits,treillis)*. Notez que vous avez également la possibilité de passer en paramètre à *convenc.m* une matrice de poinçonnage.

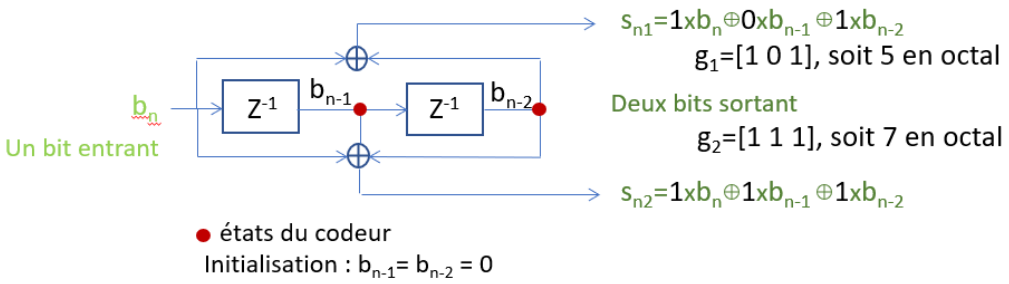


Figure 9: Codeur convolutif $(3, 1/2)$ de polynômes générateurs 5 et 7 en octal

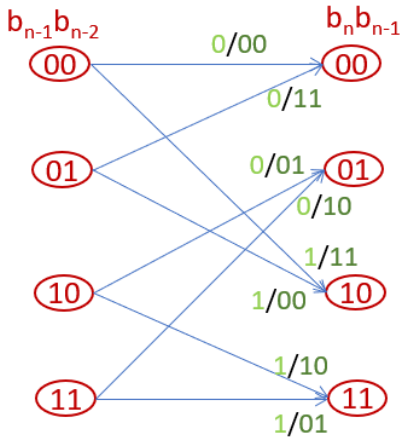


Figure 10: Treillis associé au codeur convolutif $(3, 1/2)$ de polynômes générateurs 5 et 7 en octal

```
>> treillis=poly2trellis(3,[5 7])

treillis =

  struct with fields:

     numInputSymbols: 2
    numOutputSymbols: 4
           numStates: 4
          nextStates: [4×2 double]
             outputs: [4×2 double]
```

Figure 11: Construction du treillis associé au codeur convolutif $(3, 1/2)$, de polynômes générateurs 5 et 7 en octal, avec *poly2trellis.m* de Matlab

```
>> treillis.nextStates

ans =

     0     2     On part de l'état 0 : on entre 0 on passe à l'état 0 / on entre 1 on passe à l'état 2
     0     2     On part de l'état 1 : on entre 0 on passe à l'état 0 / on entre 1 on passe à l'état 2
     1     3     On part de l'état 2 : on entre 0 on passe à l'état 1 / on entre 1 on passe à l'état 3
     1     3     On part de l'état 3 : on entre 0 on passe à l'état 1 / on entre 1 on passe à l'état 3
```

```
>> treillis.outputs

ans =

     0     3     On part de l'état 0 : on entre 0 => on sort 00, soit 0 en octal / on entre 1 => on sort 11 soit 3 en octal
     3     0     On part de l'état 1 : on entre 0 => on sort 11, soit 3 en octal / on entre 0 => on sort 00 soit 0 en octal
     1     2     On part de l'état 2 : on entre 0 => on sort 01, soit 1 en octal / on entre 1 => on sort 10 soit 2 en octal
     2     1     On part de l'état 3 : on entre 0 => on sort 10, soit 2 en octal / on entre 1 => on sort 01 soit 1 en octal
```

Figure 12: Table des états suivants et table des sorties fournie par *poly2trellis.m* de Matlab pour le code $(3, 1/2)$, de polynômes générateurs 5 et 7 en octal

5.2 Exemple de décodage du code $(3, 1/2)$

Comme le montre la figure 13, à une séquence de bits d'entrée va correspondre un chemin dans le treillis.

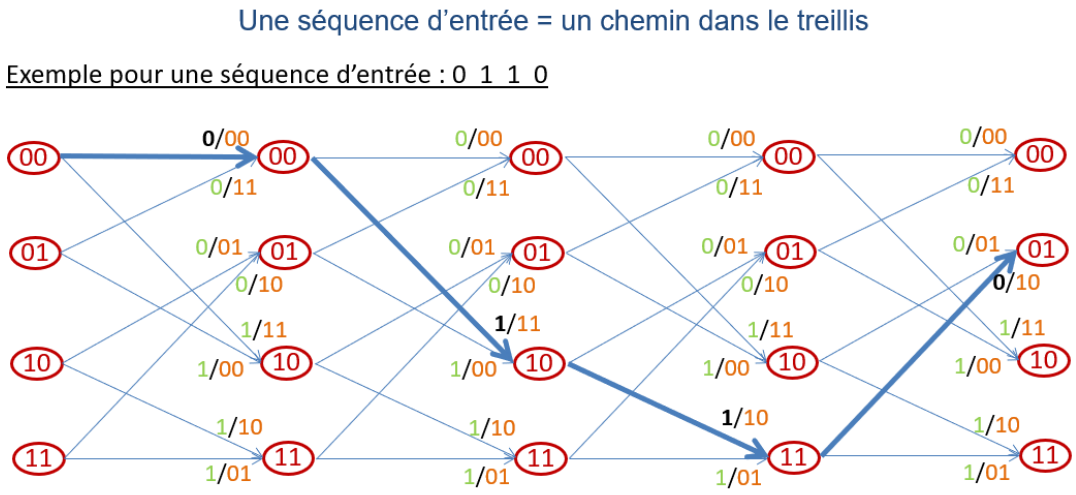


Figure 13: Exemple de chemin pour une séquence d'entrée, code $(3, 1/2)$

Si nous voulons réaliser un décodage par maximum de vraissemblance, nous devons déterminer quelle est la séquence la plus vraissemblable parmi toutes les séquences émises possibles, ce qui correspond à un nombre d'opérations à réaliser qui est prohibitif : pour une séquence de longueur N, il y a ($2^N \times$ nombre d'états) chemins possible dans le treillis, c'est-à-dire séquences possibles à tester, soit, par exemple, pour $N = 20$ et 4 états 4194304 séquences !

Il est donc nécessaire d'utiliser un algorithme et nous allons utiliser celui de Viterbi ([2]).

Deux modes de décodage sont possibles : hard et soft, selon que l'on entre dans l'algorithme des valeurs binaires, obtenues après une prise de décision, ou bien des valeurs réelles (éventuellement quantifiées).
Le principe va consister à calculer, à chaque étape du treillis, les distances (de Hamming ou euclidienne) cumulées des chemins arrivant sur chaque noeud et à ne conserver, à chaque fois, que le chemin conduisant à la plus faible.
Deux modes de décodage sont ici aussi également possible : un décodage bloc pour lequel les décisions sont prises sur l'ensemble d'un bloc de données émises et un décodage au fil de l'eau, pour lequel à chaque instant n la décision est prise sur le bit émis à l'instant $n - k$. La taille du bloc, ou bien la distance de retour k (traceback) doivent être égales à 5 à 6 fois la longueur de contrainte du code.

Le temps de calcul pour une séquence de longueur N est ainsi réduit à ($2N \times$ nombre d'états).

La figure 14 illustre l'algorithme de Viterbi sur un exemple avec un décodage soft (données réelles en entrée), si prise de décision sur le premier bit émis nous avons un décodage au fil de l'eau avec une longueur de retour de 4, si prise de décision sur l'ensemble des bits émis nous avons un décodage par bloc.

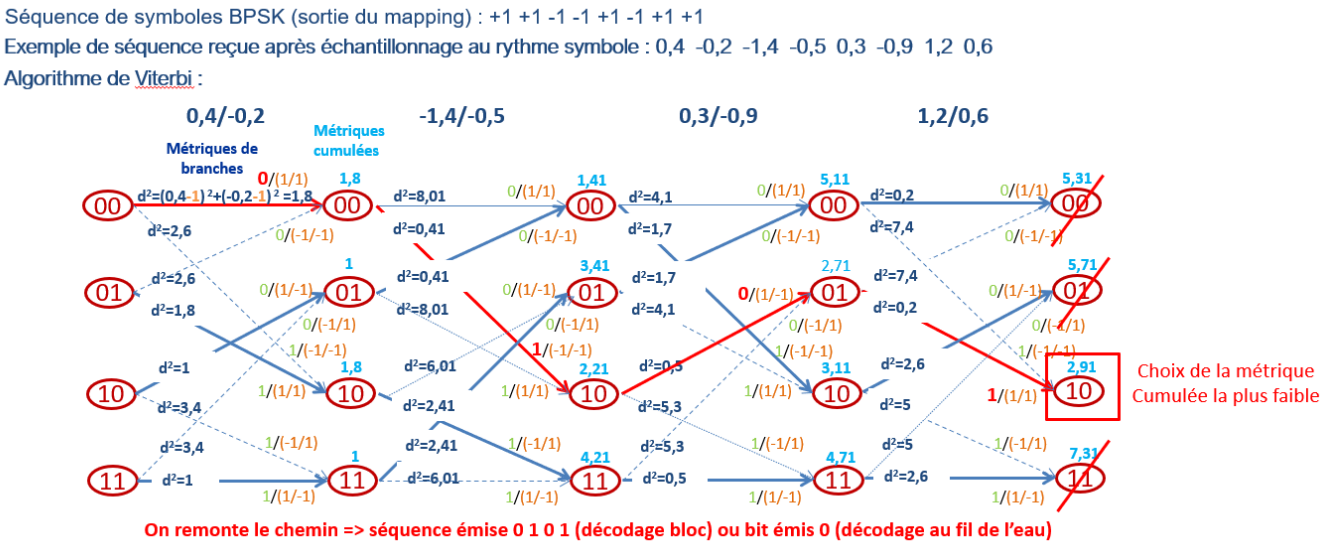


Figure 14: Exemple de décodage en utilisant l'algorithme de Viterbi pour un code $(3, 1/2)$

6 Introduction des entrelaceurs

Ajoutez à la chaine précédente les entrelaceurs définis dans la norme IEEE802.11a (voir section 2.3) et comparez les taux d'erreurs binaires obtenus sans et avec entrelacement.

7 Ajout d'un code LDPC

Afin d'évaluer l'impact d'un codage canal plus performant, vous modifierez, dans cette partie, le codage canal pour introduire un codage LDPC qui apparait dans les versions IEEE802.11n et suivantes ([3]) et pourrez comparer les courbes de TEB obtenus avec codage convolutif et avec codage LDPC.

7.1 Description des codes de la norme

Les codes LDPCs utilisés pour la standard WiFi sont des codes à structure quasi-cyclique parcimonieuse. Les codes LDPC normalisés sont des codes en bloc linéaires binaires systématiques de dimension k (taille du bloc d'information) et de longueur n (taille des mots du code). On définit ces codes à l'aide de leur matrice de parité $\mathbf{H}$ de dimension $m \times n$, où $m = n - k$. Les matrices définies étant une classe particulière de codes LDPC (codes IRA), structurées pour un codage lnéaire en temps, on peut à partir du vecteur d'information $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1}) \in \mathbb{F}_2^k$ obtenir le mot de code $\mathbf{c} = [\mathbf{u}, \mathbf{p}] \in \mathbb{F}_2^n$

$$\begin{aligned}\mathbf{c} &= (c_0, c_1, \ldots, c_{k-1}, c_k, \ldots\ldots, c_n) \\ &= (u_0, u_1, \ldots u_{k-1}, p_0, p_1, \ldots, p_{n-k-1})\end{aligned}$$

vérifiant

$$\mathbf{H} \times \mathbf{c}^T = 0.$$

Généralement les matrices construites sont de rang plein, garantissant que le rendement de codage effectif est bien $R = k/n$. Les différentes tailles et rendements supportés sont donnés par la Table1.

Rendement (R)	k (bits)	k (bits)
1/2	972	1944
1/2	648	1296
1/2	324	648
2/3	1296	1944
2/3	864	1296
2/3	432	648
3/4	1458	1944
3/4	972	1296
3/4	486	648
5/6	1620	1944
5/6	1080	1296
5/6	540	648

Table 1: Tables des tailles et rendements associés aux codes LDPC définis dans la norme 802.11n. [**?**]

Les matrices des codes LDPC du Wifi sont des matrices circulantes définies à l'aide d'une matrice de parité définie par bloc de taille $Z \times Z$, où Z est défini par le standard comme le facteur d'expansion. A chaque bloc de taille $Z \times Z$ est associée une matrice circulante (dîte matrice de permutation cyclique) ou une matrice bloc toute à zéro. A chaque bloc non nul est associé une matrice de permutation cyclique $\mathbf{P}_l, l \in [\![0, Z-1]\!]$, obtenue par permutation cyclique des colonnes de la matrice identité de taille $Z \times Z$ de l positions vers la droite. On a ainsi par définition $\mathbf{P}_0 = \mathbf{I}_Z$. Soit $\mathbf{P}_1$ la matrice circulante obtenue à partir du décalage de une position vers la droite, on vérifie alors que l'ensemble des matrices P_l peuvent être définies comme une puissance de la matrice $\mathbf{P}_1$, ie $\mathbf{P}_l = \mathbf{P}_1^k, k \in [\![0, Z-1]\!]$, les operations étant définies sur $\mathbb{F}_2$. On définit ainsi un groupe cyclique de matrice d'ordre Z. Un exemple, comme spécifié dans la norme, est donné pour $Z = 8$ par

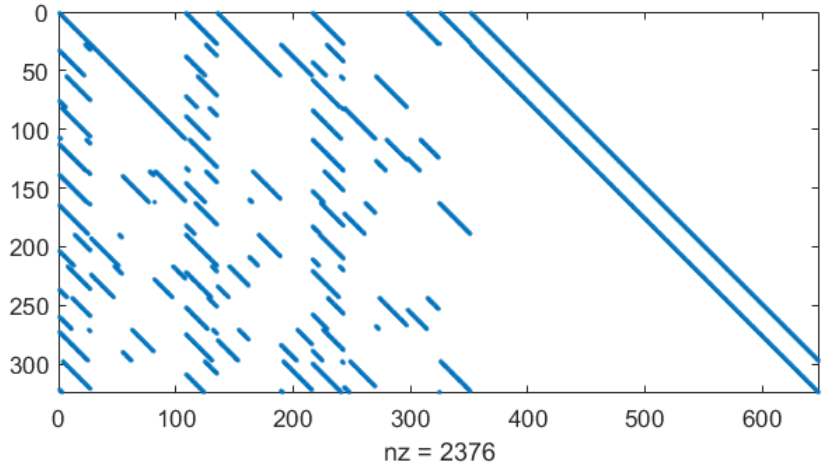
$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, P_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Les bloc sont alors référencés en utilisant l'exposant de $\mathbf{P}_1$ et les blocs tout à zéro par $-$ (ou -1). Un exemple de matrice

normalisée est donnée pour $n = 648$ bits , $R = 1/2$ et $Z = 27$ parla matrice de permutations

$$\mathbf{P} = \begin{bmatrix}
0 & - & - & - & 0 & 0 & - & - & 0 & - & - & 0 & 1 & 0 & - & - & - & - & - & - & - & - & - & - \\
22 & 0 & 0- & - & 17 & - & 0 & 0 & 12 & - & - & - & - & 0 & 0 & - & - & - & - & - & - & - & - & - \\
6 & - & 0 & - & 10 & - & - & - & 24 & - & 0 & - & - & - & 0 & 0 & - & - & - & - & - & - & - & - \\
2 & - & - & 0 & 20 & - & - & - & 25 & 0 & - & - & - & - & - & 0 & 0 & - & - & - & - & - & - & - \\
23 & - & - & - & 3 & - & - & - & 0 & - & 9 & 11 & - & - & - & - & 0 & 0 & - & - & - & - & - & - \\
24 & - & 23 & 1 & 17 & - & 3 & - & 10 & - & - & - & - & - & - & - & - & 0 & 0 & - & - & - & - & - \\
25 & - & - & - & 8 & - & - & - & 7 & 18 & - & - & 0 & - & - & - & - & - & 0 & 0 & - & - & - & - \\
13 & 24 & - & - & 0 & - & 8 & - & 6 & - & - & - & - & - & - & - & - & - & - & 0 & 0 & - & - & - \\
7 & 20 & - & 16 & 22 & 10 & - & - & 23 & - & - & - & - & - & - & - & - & - & - & - & 0 & 0 & - & - \\
11 & - & - & - & 19 & - & - & - & 13 & - & 3 & 17 & - & - & - & - & - & - & - & - & - & 0 & 0 & - \\
25 & - & 8 & - & 23 & 18 & - & 14 & 9 & - & - & - & - & - & - & - & - & - & - & - & - & - & 0 & 0 \\
3 & - & - & - & 16 & - & - & 2 & 25 & 5 & - & - & 1 & - & - & - & - & - & - & - & - & - & - & 0
\end{bmatrix}$$

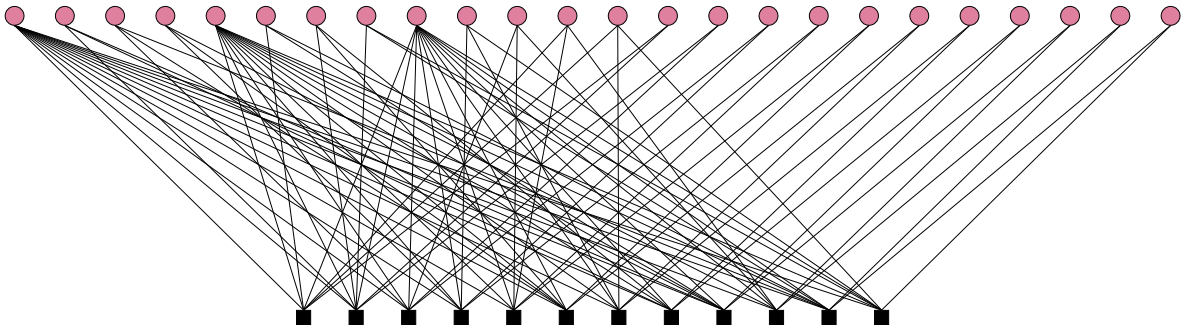
La matrice de parité est donnée par la figure suivante, où l'on voit clairement la structure quasicyclique.



La matrice de base ou protomatrice correspondante est donnée par

$$\mathbf{B} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Le protographe de taille 12×24 est alors donné par le graphe suivant



Les matrices de permutation à utiliser sont fournies en annexe.

7.2 Déclaration des configurations pour le codage

Pour déclarer une matrice de parité on utilisera le code suivant :

```
%% Permutations matrix: R=1/2, N=648, Z=27.
blockSize = 27; % taille de la ciculante ou lifting
P = [
0  -1 -1 -1  0  0 -1 -1  0 -1 -1  0  1  0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
22  0 -1 -1 17 -1  0  0 12 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1 -1;
6  -1  0 -1 10 -1 -1 -1 24 -1  0 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1;
2  -1 -1  0 20 -1 -1 -1 25  0 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1;
23 -1 -1 -1  3 -1 -1 -1  0 -1  9 11 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1;
24 -1 23  1 17 -1  3 -1 10 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1;
25 -1 -1 -1  8 -1 -1 -1  7 18 -1 -1  0 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1;
13 24 -1 -1  0 -1  8 -1  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1;
7 20 -1 16 22 10 -1 -1 23 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1;
11 -1 -1 -1 19 -1 -1 -1 13 -1  3 17 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1;
25 -1  8 -1 23 18 -1 14  9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0;
3  -1 -1 -1 16 -1 -1  2 25  5 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0
];
%% Build Parity-Check matrix
H = ldpcQuasiCyclicMatrix(blockSize,P);
```

La matrice obtenue est alors du type *sparse logical*. Pour configurer le codeur, il faut utiliser la commande suivante :

```
%% Initialization de l'objet codeur LDPC
cfgLDPCEnc = ldpcEncoderConfig(H)
```

On obtient l'objet suivant

```
cfgLDPCEnc =

  ldpcEncoderConfig with properties:

     ParityCheckMatrix: [324 648  logical]

   Read-only properties:
           BlockLength: 648
    NumInformationBits: 324
    NumParityCheckBits: 324
              CodeRate: 0.5000
```

On peut alors accèder aux attributs de l'objet par des commandes du type `cfgLDPCEnc.NumInformationBits`. L'opération de codage s'effectue avec la commande suivante :

```
%% Codage pour codes LDPC
infoBits = rand(cfgLDPCEnc.NumInformationBits,1) < 0.5; % generation des bits d'
    information
codeword = ldpcEncode(infoBits, cfgLDPCEnc);
```

7.3 Déclaration des configurations pour le décodage

Pour configurer le décodeur d'un code LDPC, il faut utiliser la commande suivante :

```
%% Initialization de l'objet d codeur LDPC
cfgLDPCDec = ldpcDecoderConfig(H);
```

On obtient l'objet suivant :

```
cfgLDPCDec =

  ldpcDecoderConfig with properties:

     ParityCheckMatrix: [324 648  logical]
             Algorithm: 'bp'

   Read-only properties:
           BlockLength: 648
    NumInformationBits: 324
    NumParityCheckBits: 324
              CodeRate: 0.5000
```

Par défaut, le décodeur utilisé sera le décodeur *Belief Propagation* ('bp'). Pour changer, l'algorithme, on utilisera, pour un algorithme 'offset-min-sum' par exemple, le code suivant :

```
%% Initialization de l'objet d codeur LDPC
cfgLDPCDec = ldpcDecoderConfig(H,'offset-min-sum');
```

On pourra comparer les différents types de décodage suivants:

1. *Belief Propagation* ('bp', BP) :dans cette version le décodeur n'est pas utilisé en mode de décodage par couche par couche;
2. *Layered Belief Propagation* ('layered-bp', L-BP) : dans cette version le décodeur BP est utilisé en mode de décodage par couche par couche; Il sera environ deux plus rapide (environ deux fois moins d'itérations pour atteindre le même taux d'erreur trame (FER)).
3. *Normalized Min-Sum Decoding* ('norm-min-sum', NMS) : dans cette version le décodeur est utilisé en mode de décodage par couche;
4. *Offset Min-Sum Decoding* ('offset-min-sum', OMS) : dans cette version le décodeur est utilisé en mode de décodage par couche;

On gardera les paramètres par défaut pour les algorithmes de type OMS et NMS. Pour les algorithmes par couche (*layered*) la taille des couches est automatiquement prise comme la taille de la dernière expansion cyclique (dans notre cas, une seule expansion de taille Z). C'est le taux de parallélisme maximal de l'algorithme, qui permet de réduire la latence du décodeur (mais pas le nombre de calculs !). Dans notre exemple, on accède à ce paramètre par la commande suivante :

```
>> cfgLDPCDec.NumRowsPerLayer

ans =

    27
```

La commande pour décoder le signal reçu est donné par exemple :

```
[APPllr,actualnumiter,syndrome] = ldpcDecode(Lch,cfgLDPCDec,itermax,OutputFormat='whole'
   ,DecisionType='soft',Termination='early');
```

Les paramètres d'entrées ici sont:

1. *Lch* : le vecteur des log-likelihood ratios (LLRs) issus des observations du canal. Ce dernier est obtenu après démodulation souple à partir des observations issues de la FFT et assumant la connaissance du canal par sous-porteuse (à minima une estimée).
2. *cfgLDPCDec* : l'objet de configuration du décodeur du code LDPC.
3. *itermax* : on définit ici le nombre maximum d'itérations de décodage;

4. *DecisionType* on définit le type de sortie du décodeur. L'argument `'soft'` permet de renvoyer les valeurs des LLRs a postériori calculés. . L'argument `'hard'` permet de ne renvoyer que les décisions dures.

5. *OutputFormat* : le type de sortie que l'on souhaite après décodage. Le format `'whole'` permet de retourner tout le mot de code. Seules les k premières entrées correspondent aux bits d'information (codes systématiques). Le format `'info'` permet quant à lui de ne retourner que les bits d'information.

6. *Termination* : L'option `Termination='early'` permet de sortir du décodeur dès que le syndrome est nul.

Les paramètres de retour sont :

1. *APPllr* : le vecteur des log-likelihood ratios (LLRs) a posteriori util à la décision si on a une configuration pour un décodage souple, `DecisionType='soft'`. Si `DecisionType='hard'`, on récupère les décisions dures (valeurs '0' ou '1').

2. *actualnumiter* : le nombre effectif d'itérations pour converger vers soit un mot de code ou le nombre d'itérations maximum.

3. *syndrome* : la valeur du syndrome, après décodage. Si on a un syndrome nul, on a effectivement convergé vers un mot de code (pas forcément celui envoyé), sinon le décodeur n'a pas convergé vers un mot de code et cette erreur est détectée.

7.4 Démodulation souple en OFDM

Le décodage itératif des codes LDPCs est par nature un algorithme à entrées souples. L'entrée du décodeur LDPC implémenté en Matlab requiert la fourniture d'un vecteur dont les entrées sont homogènes à des log-rapports de probabilité (log likelihood ratios, LLR). Ces derniers sont des LLRs calculés à partir d'un critère de démodulation MAP bit exact ou approximé.

Démodulation souple : cas gaussien

Pour une modèle gaussien, nous avons

$$y[n] = x[n] + b[n],$$

où $y[n]$ est le symbole bruité reçu, $x[n] \in \mathcal{X}$ est un symbole de la constellation $\mathcal{X} \in \mathcal{C}$ (symboles supposés i.i.d.) et $b[n] \sim \mathcal{CN}(0, \sigma^2)$, avec σ^2 la variance du bruit blanc complexe gaussien. On note $\underline{\mathbf{b}} = [b_1, b_2, \cdots, b_m] \in \mathbb{F}_2^m$ le m-uplet représentant le label m-binaire associé aux symboles de la constellation $\mathcal{X}$, avec $|\mathcal{X}| = 2^m$. L'opération de mapping est représentée par $\mathcal{M} : \mathbb{F}_2^m \to \mathcal{X}$, et telle que $x = \mathcal{M}(\underline{\mathbf{b}})$. On peut également noter $\mathcal{M}_i^{-1} : \mathcal{X} \to \mathbb{F}_2$, l'opération de récupération du label binaire de la i-ième coordonnée binaire de $\underline{\mathbf{b}}$ pour un symbole donné, avec $b_i = \mathcal{M}_i^{-1}(x)$. L'expression 'exacte' du log-rapport de probabilité associé à une détection binaire avec critère MAP est donnée pour la i-ème coordonnée par :

$$\begin{aligned}\mathbb{L}(b_i[n]) &= \log\left(\frac{\Pr(b_i[n]=0 \mid y[n])}{\Pr(b_i[n]=1 \mid y[n])}\right) \\ &= \log\left(\frac{\sum_{x\in\mathcal{X}_0^i} e^{-\frac{1}{\sigma^2}|y[n]-x|^2}}{\sum_{x\in\mathcal{X}_1^i} e^{-\frac{1}{\sigma^2}|y[n]-x|^2}}\right)\end{aligned}$$

où $\mathcal{X}_b^i = \{x \in \mathcal{X} \mid b_i = b\} = \{x \in \mathcal{X} \mid \mathcal{M}_i^{-1}(x) = b\}, b \in \{0, 1\}$. En Matlab, la démodulation souple est accessible directement à partir de fonctions de démodulation intégrées, calibrées pour le cas gaussien.

La version approximée du critère MAP est donnée par

$$\mathbb{L}(b_i[n]) = -\frac{1}{\sigma^2}\left(\min_{x\in\mathcal{X}_0^i} |y[n]-x|^2 - \min_{x\in\mathcal{X}_1^i} |y[n]-x|^2\right)$$

Pour des modulations de type QAM, on a la syntaxe suivante pour une modulation d'ordre M :

```
%% Demodulateur QAM
 LLR = qamdemod(y,M, OutputType='llr', ...
            UnitAveragePower=true, ...
            NoiseVariance=noiseVar);
```

Par défaut, le mapping utilisé est de type Gray. L'argument `OutputType='llr'` permet de spécifier que le vecteur de sortie renvoyé est homogène à des LLRs dont l'expression 'est exacte'. Cet argument est à changer par `OutputType='approxllr'` pour une approximation de type max – log MAP. `UnitAveragePower=true` considère une constellation de puissance unitaire. Le paramètre `NoiseVariance=noiseVar` correspond à la variance du bruit complexe.

Warning

Attention le paramètre `NoiseVariance` est à actualiser pour chaque rapport signal-à-bruit simulé.

A noter

On associera préférentiellement une démodulation de type `OutputType='llr'` avec un décodage de type `'bp'` ou `'layered-bp'` et une démodulation de type `OutputType='approxllr'` avec une décodeur de type `'offset-min-sum'` ou `'normalized-min-sum'` afin de considérer des types de variables échangées homogènes.

Bien sûr, ces fonctions de démodulations supposent l'utilisation de leur fonction duale à l'émetteur pour assurer la cohérence des opérations de mapping des bits vers les symboles de modulations de type QAM. La syntaxe est alors donnée par

```
%% Modulateur QAM
txSig = qammod(dataBits,M,InputType='bit',UnitAveragePower=true);
```

Par défaut, le mapping utilisé est de type Gray. L'argument `InputType='bit'` permet de spécifier que le vecteur d'entrée est composé de bits et `UnitAveragePower=true` garantit une constellation de puissance unitaire.

Démodulation souple : extension au cas OFDM

L'approche précédente doit être adaptée pour le cas OFDM. L'utilisation d'une modulation OFDM conjointement avec l'utilisation d'un préfixe cyclique permet de transformer en réception un canal sélectif en fréquence en canal sélectif en temps, supposé stationnaire sur au moins un symbole OFDM. Ainsi, pour un symbole de taille N_{fft}, le modèle discret équivalent après FFT de réception est donné par

$$Y[k] = H[k]X[k] + B[k], \forall k \in [\![0, N_{\text{fft}-1}]\!]$$

où $Y[n]$ est le symbole bruité reçu après FFT, $X[n] \in \mathcal{X}$ est un symbole de la constellation $\mathcal{X} \in \mathcal{C}$ (symboles supposés i.i.d.), $H[k]$ sont les coefficients de la transformée de Fourier discrète de la réponse impulsionnelle du canal discret équivalent $h[n]$, et $B[n] \sim \mathcal{CN}(0, \sigma^2)$ après FFT, avec σ^2 la variance du bruit blanc complexe gaussien. L'expression 'exacte' du log-rapport de probabilité associé à une détection binaire avec critère MAP est donnée pout tout $i \in [\![1, m]\!]$ et $k \in [\![0, N_{\text{fft}-1}]\!]$ par :

$$\begin{aligned}
\mathbb{L}(b_i[k]) &= \log\left(\frac{\Pr(b_i[k]=0 \mid Y[k], H[k])}{\Pr(b_i[k]=1 \mid Y[k], H[k])}\right) \\
&= \log\left(\frac{\sum_{X\in\mathcal{X}_0^i} e^{-\frac{1}{\sigma^2}|Y[k]-H[k].X|^2}}{\sum_{X\in\mathcal{X}_1^i} e^{-\frac{1}{\sigma^2}|Y[k]-H[k].X|^2}}\right) \\
&= \log\left(\frac{\sum_{X\in\mathcal{X}_0^i} e^{-\frac{|H[k]|^2}{\sigma^2}|Y[k]/H[k]-X|^2}}{\sum_{X\in\mathcal{X}_1^i} e^{-\frac{|H[k]|^2}{\sigma^2}|Y[k]/H[k]-X|^2}}\right) \\
&= \log\left(\frac{\sum_{X\in\mathcal{X}_0^i} e^{-\frac{1}{\sigma_k^2}\left|\tilde{Y}[k]-X\right|^2}}{\sum_{X\in\mathcal{X}_1^i} e^{-\frac{1}{\sigma_k^2}\left|\tilde{Y}[k]-X\right|^2}}\right)
\end{aligned}$$

On en déduit que la détection dans le cas OFDM se ramène facilement au cas gaussien en considérant le modèle d'observation modifié suivant :

$$\tilde{Y}[k] = W[k]Y[k] = X[k] + \tilde{B}[k],$$

avec $W[k] = 1/H_k$ et $\tilde{B}[k] \sim \mathcal{CN}(0, \sigma_k^2)$, avec $\sigma_k^2 = \sigma^2/|H[k]|^2$. On reconnait alors $W[k] = W_{zf}[k]$, l'égaliseur 'zero-forcing' (ZF) scalaire. Le modèle d'observation modifié est donc équivalent à celui obtenu après application d'un égaliseur ZF. L'application de la détection au sens du MAP sur le modèle équivalent en sortie d'égaliseur ZF donnera donc les mêmes performances que le modèle d'observation direct (l'égalisation ZF est équivalente dans le cas scalaire à retirer un biais d'"estimation" dû à la présence du terme $H[k]$).

On peut alors en déduire une méthode simple d'utilisation du démodulateur souple de Matlab, implémenté pour le cas gaussien : il suffit de d'appliquer l'égalisation ZF, puis de considérer la variance modifiée σ_k^2 pour la démodulation souple. On aura une implémentation du type [1]:

[1] exemple non contractuel

```
%% Demodulateur QAM OFDM
X_hat=Y./H;
noiseVarEff=noiseVar*1./abs(H).^2;
LLR = qamdemod(X_hat,M, OutputType='llr', ...
            UnitAveragePower=true, ...
            NoiseVariance=noiseVarEff);
```

On a dans cet exemple utilisé le fait que l'on pouvait vectoriser la démodulation souple en donnant un vecteur pour la variance de bruit. Ici, on suppose donc que l'on démodule mot de code par mot de code. Reportez-vous à la documentation pour plus d'information.

A noter

On peut également montrer que pour un récepteur MMSE, on aura le même résultat.En effet, pour un récepteur MMSE, on a

$$\forall k,\ W_{\text{mmse}}[k] = \frac{H[k]^*}{|H[k]|^2 + \text{snr}^{-1}}.$$

On en déduit que le modèle après égaliseur est donné par

$$\begin{aligned}\hat{X}_{\text{mmse}}[k] &= W_{\text{mmse}}[k]Y[k] \\ &= \frac{|H[k]|^2}{|H[k]|^2 + \text{snr}^{-1}1}X[k] + \frac{H[k]^*}{|H[k]|^2 + \text{snr}^{-1}}B[k] \\ &= \alpha[k]X[k] + \tilde{B}[k].\end{aligned}$$

On a donc une estimation biaisée de biais $\alpha[k]$ et $\tilde{B}[k] \sim \mathcal{CN}(0, \sigma^2|W_{\text{mmse}}[k]|^2)$. Pour la démodulation au sens du MAP bit, on doit prendre en compte ce biais et la variance modifiée du bruit. En utilisant les mêmes arguments que précédemment, en cherchant à enlever ce biais, le modèle d'obserbation débiaisé donne

$$\begin{aligned}\tilde{Y}[k] &= \hat{X}_{\text{mmse}}[k]/\alpha[k] \\ &= X[k] + B[k]/H[k]\end{aligned}$$

qui est exactement le même modèle que le cas avec égaliseur ZF! On a donc pour une démodulation optimale au sens du MAP bit des performances identiques sur le modèle direct, et pour les stratégies de détections MAP après égaliseur ZF ou MMSE (Les snr équivalents pour les modèles débiaisés sont équivalents). Ceci est vrai car nous sommes dans un modèle scalaire. Le même résultat serait obtenu pour un filtre scalaire ML $W_{\text{ml}}[k] = H[k]^*$ (et pour tout type de détection après filtrage du type $W_{\text{x}}[k] = \beta H[k]^*$).

8 Estimation du canal de propagation en utilisant les porteuses pilotes

En pratique, le canal d'estimation est estimé en utilisé les 4 porteuses pilotes dédiées. Vous mettrez en place cette estimation du canal de propagation, en supposant connues du récepteur les informations transportées par les pilotes, et pourrez comparer les TEBs obtenus, losrque le canal de propagation est supposé parfaitement connu et lorsqu'il est estimé à partir des porteuses pilotes.

9 Notation - Consignes pour les rendus et l'oral

La note du projet sera composée d'une note d'oral, d'une note de rapport et d'une note d'implantation. La note d'oral comptera pour 50% de la note finale, tandis que la note rapport + implantation fournira les 50% restants.

9.1 Soutenance orale

La soutenance orale va permettre d'évaluer :

1. que vous êtes bien les auteurs des programmes MATLAB que vous présenterez. Vous devrez être en mesure d'expliquer ce que fait chaque ligne de code, d'expliquer ce que font les fonctions que vous utilisez et de justifier leur paramétrage, par rapport au système à implanter ou par rapport à des concepts théoriques de communications numériques, OFDM et codage canal.

2. que vous savez évaluer les performances des chaînes de transmission numériques. Vous devrez être en mesure d'analyser les courbes de performance des différentes chaînes de transmissions que vous avez simulées et de comparer ces courbes en justifiant leurs positions relatives. Ces justifications doivent être fondées sur une analyse des blocs constituant chacune des chaînes de transmissions simulées. Savoir analyser un bloc d'une chaîne de transmission, c'est savoir expliquer son fonctionnement, justifier sa présence dans la chaîne, savoir expliquer ses propriétés, ses limites.

Nous vous conseillons de réaliser le rapport au fur et à mesure des séances car il vous permettra de vous questionner sur les points que nous aborderons lors de l'oral. Vous pouvez venir à l'oral avec votre rapport ou une première ébauche de votre rapport.
Soyez également au clair sur les fichiers Matlab à lancer pour nous montrer telle ou telle partie du projet.

9.2 Rapport

Le rapport va permettre d'évaluer le point 2 de la soutenance orale, mais à l'écrit.

Comme tout rapport, il doit comporter : un sommaire, une introduction présentant les objectifs du travail, une conclusion synthétisant les principaux résultats obtenus et une bibliographie comprenant les références éventuellement utilisées, notamment pour expliquer vos résultats. On peut y ajouter une table des illustrations.

Les équations doivent être réalisées avec un éditeur d'équation.

Lorsque vous commentez une figure vous devez y faire référence dans votre texte : par exemple "comme le montre la figure 1, ..."

Les figures inclues dans vos rapports doivent être lisibles et toutes comporter un titre (utiliser *title* sous Matlab) , des labels sur leurs axes (utiliser *xlabel* et *ylabel* sous matlab) ainsi qu'une légende si plusieurs courbes sont tracées sur la même figure (utiliser *legend* sous matlab).

Toutes vos explications/justifications doivent utiliser les bons termes techniques (provenant des cours/TDs/TPs, des livres/sites consultés et cités), pas d'à peu près. "En gros" est à proscrire...

Attention votre rapport doit être relu, éventuellement passé au correcteur orthographique et grammatical.

9.3 Codes

L'examen des codes MATLAB va permettre d'évaluer :

1. que vous savez programmer dans ce langage. Une attention particulière sera portée sur l'utilisation pertinente et explicite des noms pour les fichiers, constantes, variables, fonction, ?, sur la présence de commentaires, sur l'utilisation et la déclaration de constantes en début de programme.

2. que vous savez adapter la programmation aux contraintes particulières de la simulation de chaînes de transmission numérique, notamment en écrivant des commentaires utilisant la terminologie adéquate des enseignements de communications numériques et de codage canal.

Les fichiers .m fournis devront porter des noms significatifs et permettre d'exécuter les différentes étapes demandées dans le projet, sans avoir à ajouter des commentaires pour supprimer certaines parties... Si besoin, vous pouvez fournir un mode d'emploi dans le rapport pour savoir ce qui doit être lancé pour réaliser les différentes fonctions implantées.

10 Annexes

10.1 Matrices de permutations pour $N = 648$ bits codés

```
%% Permutations matrix: R=1/2, N=648, Z=27.
blockSize = 27; % taille de la ciculante ou lifting
P = [
 0 -1 -1 -1  0  0 -1 -1  0 -1 -1  0  1  0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
22  0 -1 -1 17 -1  0  0 12 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1 -1;
 6 -1  0 -1 10 -1 -1 -1 24 -1  0 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1 -1;
 2 -1 -1  0 20 -1 -1 -1 25  0 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1 -1;
23 -1 -1 -1  3 -1 -1 -1  0 -1  9 11 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1 -1;
24 -1 23  1 17 -1  3 -1 10 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1 -1;
25 -1 -1 -1  8 -1 -1 -1  7 18 -1 -1  0 -1 -1 -1 -1 -1  0  0 -1 -1 -1 -1;
13 24 -1 -1  0 -1  8 -1  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1 -1;
 7 20 -1 16 22 10 -1 -1 23 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1 -1;
11 -1 -1 -1 19 -1 -1 -1 13 -1  3 17 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0 -1;
25 -1  8 -1 23 18 -1 14  9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0  0;
 3 -1 -1 -1 16 -1 -1  2 25  5 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  0
];

%% Permutations matrix: R=3/4, N=648, Z=27.
blockSize = 27; % taille de la ciculante ou lifting
P = [
16 17 22 24  9  3 14 -1  4  2  7 -1 26 -1  2 -1 21 -1  1  0 -1 -1 -1 -1;
25 12 12  3  3 26  6 21 -1 15 22 -1 15 -1  4 -1 -1 16 -1  0  0 -1 -1 -1;
25 18 26 16 22 23  9 -1  0 -1  4 -1  4 -1  8 23 11 -1 -1 -1  0  0 -1 -1;
 9  7  0  1 17 -1 -1  7  3 -1  3 23 -1 16 -1 -1 21 -1  0 -1 -1  0  0 -1;
24  5 26  7  1 -1 -1 15 24 15 -1  8 -1 13 -1 13 -1 11 -1 -1 -1 -1  0  0;
 2  2 19 14 24  1 15 19 -1 21 -1  2 -1 24 -1  3 -1  2  1 -1 -1 -1 -1  0
];
```

11 Références

[1] : IEEE Std 802.11a-1999, IEEE Standard for Information Technology— Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks—Specific Requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHZ Band.

[2] : A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems", IEEE Transactions on Communication Technology, vol. 19, no. 5, pp. 751–772, Oct. 1971.

[3] : IEEE Std 802.11™-2020, IEEE Standard for Information Technology— Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks—Specific Requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Revision of IEEE Std 802.11-2016.

[4] : Lien vers les cours de communications numériques de 1A

[5] : Lien vers le cours OFDM de 2A

[6] : Lien vers le cours de codage canal de 2A