Conception de Codes LDPC avec l'Algorithme RandPEG Evitant les Trapping Sets Dominants

Madiagne DIOUF[1,2], David DECLERCQ[1], Samuel OUYA[2], Bane VASIC[3]

[1]ETIS ENSEA/UCP/CNRS UMR-8051, 95014 Cergy-Pontoise, (France),

[2]LIRT Ecole Supérieure Polytechnique, Univ. Cheikh Anta DIOP, Dakar, (Senegal)

[3]Dept. of Electrical and Computer Engineering, Univ. of Arizona Tucson, AZ 85721, (U.S.A.)

madiagne.diouf@ensea.fr, declercq@ensea.fr
samuel.ouya@gmail.com, vasic@ece.arizona.edu

Résumé – Dans cet article, nous proposons une méthode prédictive de construction de codes LDPC de poids colonne régulier $d_v = 3$ et de girth $g = 8$ tel que leurs graphes de Tanner contiennent un nombre minimum de "trapping-sets". Cette construction est basée sur des améliorations de l'algorithme "Progressive Edge-Growth (PEG)". Nous montrons comment détecter un "trapping-set" $(5, 3)$ et $(6, 4)$ dans l'arbre de calcul d'un noeud variable lors de la création d'une nouvelle branche. Une caractérisation précise et rigoureuse des "trapping sets" $(5, 3)$ et $(6, 4)$ est donnée. Nous proposons une modification sur l'algorithme "Randomized Progressive Edge-Growth" (RandPEG) [1] qui consiste à définir une nouvelle fonction de coût permettant de construire des codes LDPC de poids colonne régulier $d_v = 3$, de girth 8 sans (5,3) et minimisant les (6,4). Nous présentons les contructions et les résultats de performances dans le cas des codes LDPC quasi-cycliques (QC-LDPC) montrant la supériorité de notre conception de codes.

Abstract – In this paper, we propose a predictive method to construct regular column-weight-three LDPC codes with girth $g = 8$ so that their Tanner graphs contain a minimum number of small trapping sets. Our construction is based on improvements of the Progressive Edge-Growth (PEG) algorithm. We first show how to detect the smallest trapping sets $(5, 3)$ and $(6, 4)$ in the computation tree spread from variable nodes during the edge assignment. A precise and rigorous characterization of trapping sets $(5, 3)$ and $(6, 4)$ are given, and we then derive a modification of the Randomized Progressive Edge-Growth (RandPEG) algorithm [1] to take into account a new cost function that allows to build regular column-weight $d_v = 3$, girth 8 LDPC codes free of (5,3) and with a minimization of (6,4). We present the construction and the performance results in the context of quasi-cyclic LDPC (QC-LDPC) codes that show the superiority of our design codes.

1 Introduction

La conception de codes LDPC binaires avec un faible plancher d'erreurs est encore un problème considérable non entièrement résolu dans la littérature. Pour les canaux bruités, le plancher d'erreur des décodeurs LDPC est dû à la présence de certaines topologies dans le graphe de Tanner [2] tel que les cycles et les "trapping-set". Dans ce papier, nous fournissons une solution pour construire des codes LDPC utilisant une version modifiée de l'algorithme "Progressive Edge-Growth" (PEG), avec pour objectif d'éviter les structures les plus nocives dans le graphe de Tanner du code LDPC.

Dans sa version originale, l'algorithme PEG [3] est un algorithme "greedy" qui est proposé pour maximiser la taille du plus petit cycle local appelé "girth" locale du graphe de Tanner. Des améliorations de l'algorithme PEG ont été proposées pour éviter les faibles "stopping-sets" [4], ou minimiser le nombre de cycles [1] avec une généralisation non-greedy.

Cependant, il n'existe pas encore de méthode basée sur l'algorithme PEG pour détecter et éviter les "trapping-sets". Or la littérature relevant des décodeurs LDPC révèle que les problèmes de plancher d'erreurs sont principalement causés par les petits trapping sets, nous nous concentrons dans ce papier sur les plus petits trapping-sets que l'on peut considérer pour une girth donnée c'est-à-dire les trapping-sets $(5, 3)$ et $(6, 4)$ pour les codes LDPC de poids colonne régulier $d_v = 3$ avec un girth $g = 8$ [5]. La première étape de notre travail est de fournir une caractérisation de la façon dont des trapping-sets peuvent être détectés dans l'arbre de calcul qui est utilisé dans l'algorithme PEG. Avec cette caractérisation, il est possible de prédire quels sont les candidats qui créeront effectivement des trapping sets. Nous introduisons une fonction coût dans l'algorithme Randomized-PEG [1] de façon à ce que les trapping-sets $(5, 3)$ soient complétement supprimés et le nombre de trapping-sets $(6, 4)$ minimisé.

En utilisant cette approche, nous pouvons construire des codes LDPC sans petits trapping-sets pour une girth donnée, généralisant les idées présentées dans [1, 4]. Nous nous concentrons sur la conception de codes LDPC quasi-cycliques (QC-LDPC), qui sont les codes LDPC les plus utilisés dans les applications pratiques.

2 Préliminaires et Notations

Soit G le graphe de Tanner d'un code LDPC binaire $\mathcal{C}$(N,K) de rendement $R = K/N$. Ce graphe est constitué d'un ensemble de N noeuds variables V et d'un ensemble de M noeuds de contrôle C. Deux noeuds sont voisins s'il existe une branche entre eux. Le degré d'un noeud dans G est son nombre de voisins dans G. Un code $\mathcal{C}$ representé par le graphe G est dit de poids colonne régulier d_v si tous les noeuds variables dans V

ont le même degré d_v. Un chemin dans G est une séquence finie de noeuds (variables ou contrôle) distincts $u_0, \ldots, u_l$ tel que u_{i-1} and u_i sont des voisins pour $1 \leq i \leq l$. Deux chemins sont distincts s'ils diffèrent au moins d'un noeud. Un l-cycle ou cycle de longueur l dans G une suite alternative de noeud variable et noeud de contrôle $u_0, \ldots, u_l$ avec $u_0 = u_l$. Donc dans un graphe biparti G, la longueur d'un cycle l est paire. La "girth" g de G est la longueur du plus petit cycle dans G. $\mathcal{C}_{d_v,g}$ est l'ensemble des codes de poids colonne régulier d_v avec un girth g.

Nous rappelons la définition d'un "trapping set" comme défini initialement par Richardson dans [2]. *Pour un décodeur donné d'entrée* $\mathbf{y}$*, un* **trapping set** *(TS) pour un décodeur itératif dénoté par* $\mathbf{T}(\mathbf{y})$ *est un ensemble non-vide de noeuds variables dans* G *qui n'est pas corrigé à la fin du nombre d'itérations donné.* Une notation commune utilisée pour décrire un TS est (a, b), où $a = |\mathbf{T}|$, et b est le nombre de noeuds de contrôle de degré impair dans le sous-graphe formé par l'ensemble des noeuds variables $\mathbf{T}$. Soit $\mathcal{T}(a, b)$ qui dénote le graphe biparti associé avec le trapping set (a, b) , où a est le nombre de noeuds variables et b est le nombre de noeuds de contrôle de degré impair présent dans le graphe. Un graphe G contient un (a, b) de type $\mathcal{T}$ s'il existe un sous ensemble de noeuds variables $\mathbf{T}$ dans G dont le sous graphe induit est isomorphe à $\mathcal{T}(a, b)$. Un TS est dit *élémentaire* si $\mathcal{T}$ contient seulement des noeuds de contrôle de degré un et/ou deux sinon le TS est non-élémentaire. Dans ce papier, nous nous limitons sur les trapping sets élémentaires, puisqu'ils sont connu comme étant les dominants dans le plancher d'erreur [2, 6]. La notation (a, b) n'est pas suffisante pour décrire la topologie d'un trapping set car il peut y avoir plusieurs graphes non-isomorphes avec a noeuds variables et b noeuds de contrôle de degré impair. Ainsi nous utilisons la notation qui énumère les différents cycles dans le sous graphe [7]. Soit $\mathcal{T}$ qui contient g_{2k} cycles de longueur $(2k)$, où $k \geq 2$, alors le trapping set associé à $\mathcal{T}$ est dit de type $(a, b, \prod_{k\geq 2}(2k)^{g_{2k}})$. Une méthode alternative de description précise des trapping set est décrite dans [6].

Une méthode très éfficace pour construire un graphe de Tanner ayant une large girth est donné en utilisant de l'algorithme "Progressive-Edge-Growth (PEG)" [3]. Dans l'algorithme PEG, la création d'une nouvelle branche pour un noeud variable v se base sur l'expansion du sous graphe de v jusqu'à la profondeur k comme le montre la Figure 1, il est aussi appelé arbre des voisins ou encore arbre de calcul. C'est un arbre avec $2(k + 1)$ niveaux, indexé entre $[0, 2k + 1]$, où le 0^{eme} niveau consiste seulement au noeud racine v, les niveaux pairs contiennent uniquement des noeuds variables représentés par $\circ$ tandis que les niveaux impairs contiennent uniquement les noeuds de contrôle représentés par $\blacksquare$.

La profondeur d'arrêt de l'arbre k dépends du girth ciblé [1], et le sous graphe est appellé *arbre de profondeur-k* et est noté T_v. Nous notons l'ensemble des noeuds de contrôle voisins présents dans l'arbre de profondeur-k par $\mathcal{M}_v^k$, et $\overline{\mathcal{M}}_v^k$ l'ensemble complémentaire. Un noeud $w \in T_v(G)$ est dit un *ascendant* du noeud node $u \in T_v(G)$ s'il existe un chemin de la racine v au noeud u qui traverse le noeud w. L'ensemble des noeuds variable ascendants du noeud u dans T_v est noté $\mathcal{A}(u)$. Un noeud $w \in T_v$ est dit un *parent* du noeud $u \in T_v$ si w est directement connecté à u sur le chemin traversant depuis la racine v. Il est clair que $w \in \mathcal{N}_u$. Lors de l'expansion de l'arbre de profondeur-k, différentes copies d'un même noeud variable peuvent apparaître à différents niveaux sur T_v. Soit T_v qui contient m copies du noeud u, nous noterons ces copies par $u_i^{l_i}$, $1 \leq i \leq m$, où l_i est le niveau de la i^{eme} copie.

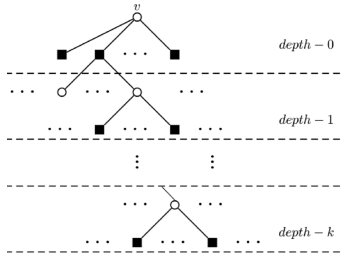


FIGURE 1 – Sous graphe étalé depuis le noeud variable v : *arbre de profondeur-k*

3 Caractérisation et Critère de détection des Trapping Sets $(5, 3)$ et $(6, 4)$

3.1 Caractérisation des Trapping Sets $(5, 3)$ et $(6, 4)$

Dans le papier, nous considérons uniquement les codes $\mathcal{C}_{d_v,g}$, avec $d_v = 3$ et $g = 8$. Un trapping set peut être considéré comme une combinaison de plusieurs cycles, et les TS (a, b) sont différenciés par le nombre de cycles distincts qui les forme. Un trapping set élémentaire est connu comme étant dominant dans le plancher d'erreur [2, 6] et parmi eux les plus nocifs sont les trapping set avec un nombre de noeuds variables a faible. Pour $\mathcal{C}_{3,8}$, les $(5, 3)$ et $(6, 4)$ sont les trapping sets dominants et ils sont formés de combinaisons de cycles. Nous donnons des caractérisations des trapping sets $(5, 3)$ et $(6, 4)$ et ces caractérisations sont utilisés pour définir les critères de détections de trapping sets dans l'arbre de profondeur-k.

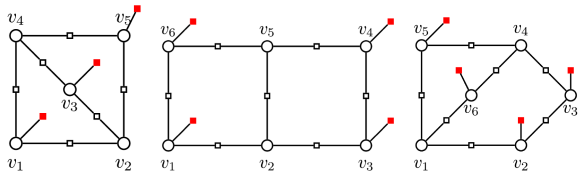
Caractérisation 1 *Un trapping set élémentaire* $(5, 3, 8^3)$ *peut être caractérisé comme une combinaison de deux 8-cycles ayant exactement trois noeuds variables en commun.*

Caractérisation 2 *Un trapping set élémentaire de type* $(6, 4, 8^2, 12^1)$ *peut être caractérisé comme une combinaison de deux 8-cycles ayant exactement deux noeuds variables en commun.*

Caractérisation 3 *Un trapping set élémentaire de type* $(6, 4, 8^1, 10^2)$ *peut être caractérisé comme une combinaison de deux 10-cycles ayant exactement quatre noeuds variables en commun.*

Ces topologies dont nous nous intéressons sont réprésentées dans les figures Fig. 2(a) à Fig. 2(c). Fig. 2(a) montre un $\mathcal{T}\left(5, 3, 8^3\right)$ composé de trois 8-cycles dont les ensembles des noeuds variables sont :$\{v_1, v_2, v_3, v_4\}$, $\{v_2, v_3, v_4, v_5\}$, et $\{v_1, v_2, v_4, v_5\}$, ces ensembles pris deux à deux ont trois noeuds

variables en commun. De manière similaire, la Fig. 2(b) montre un $\mathcal{T}\left(6,4,8^2,12^1\right)$ où nous avons deux 8-cycles dont les ensembles des noeuds variables sont :$\{v_1,v_2,v_5,v_6\}$ et $\{v_2,v_3,v_4,v_5\}$, ces deux ensembles ont v_2 et v_5 en commun. Pour $\mathcal{T}\left(6,4,8^1,10^2\right)$ représenté par la Fig. 2(c), $\{v_1,v_2,v_3,v_4,v_5\}$ et $\{v_1,v_2,v_3,v_4,v_6\}$ sont les ensembles des noeuds variables des deux 10-cycles où $\{v_1,v_2,v_3,v_4\}$ sont les noeuds variables en commun.



(a) (b) (c)

FIGURE 2 – $\mathcal{T}(5,3,8^3)$; $\mathcal{T}(6,4,8^2,12)$; $\mathcal{T}(6,4,8^1,10^2)$ respectivement

3.2 Critère de détection de trapping set (5,3) and (6,4) avec l'algorithme PEG

Dans l'algorithme PEG, les cycles peuvent être facilement prédits selon le niveau d'apparition de chaque noeud de contrôle dans l'arbre de profondeur-k. Ce pendant la détection des structures complexes tel que les TS est plus ennuyeux. Dans l'arbre de profondeur-k, un TS peut être prédit en utilisant le nombre de copies d'un noeud de contrôle avec leurs niveaux d'apparition et le nombre de noeuds variables en commun sur les différents chemins $v,.....,c_i^{l_i}$ dans T_v. Les théorèmes suivant décrivent comment détecter les trapping sets (5,3) et (6,4) dans T_v.

Théorème 1 *Soit $\mathcal{C}_{3,8}$, pour tout noeud variable v une nouvelle trapping-set $(5,3,8^3)$ est détectée dans T_v si et seulement si il existe au moins 2 copies du noeud de contrôle c denoté par c_1^7,c_2^7 avec le même* **parent**, *pour lesquels le chemin $v,...,c_1^7$ et le chemin $v,....,c_2^7$ dans T_v ont trois noeuds variables en commun (ie. $|\mathcal{A}(c_1^7)\bigcap\mathcal{A}(c_2^7)|=3$).*

Théorème 2 *Soit $\mathcal{C}_{3,8}$, pour tout noeud variable v un trapping-set $(6,4,8^2,12^1)$ est détecté dans T_v si et seulement si nous avons un des deux cas :*

1. *il existe au moins 2 copies du noeud de contrôle c denoté par c_1^7,c_2^{11} avec le même parent, pour lesquels le chemin $v,...,c_1^7$ et le chemin $v,....,c_2^{11}$ dans T_v ont quatre noeuds variables en commun (ie. $|\mathcal{A}(c_1^7)\bigcap\mathcal{A}(c_2^{11})|=4$).*
2. *il existe au moins 2 copies du noeud de contrôle c denoté par c_1^7,c_2^7 avec le même parent, pour lesquels le chemin $v,...,c_1^7$ et le chemin $v,....,c_2^7$ dans T_v ont deux noeuds variables en commun (ie. $|\mathcal{A}(c_1^7)\bigcap\mathcal{A}(c_2^7)|=2$).*

Théorème 3 *Soit $\mathcal{C}_{3,8}$, pour tout noeud variable v un trapping-set $(6,4,8^1,10^2)$ est détecté dans T_v si et seulement nous avons un des deux cas :*

1. *il existe au moins 2 copies du noeud de contrôle c dénoté par c_1^7,c_2^9 avec le même parent, pour lesquels le chemin $v,...,c_1^7$ et le chemin $v,....,c_2^9$ dans T_v ont trois noeuds variables en commun (ie. $|\mathcal{A}(c_1^7)\bigcap\mathcal{A}(c_2^9)|=3$).*
2. *il existe au moins 2 copies du noeud de contrôle c dénoté par c_1^9,c_2^9 avec le même parent, pour lesquels le chemin $v,...,c_1^9$ et le chemin $v,....,c_2^9$ dans T_v ont quatre noeuds variables en commun (ie. $|\mathcal{A}(c_1^9)\bigcap\mathcal{A}(c_2^9)|=4$).*

Les preuves sont omises par manque de place dans cet article, mais seront reportées dans un article long.

4 Conception de l'Algorithme utilisant le RandPEG

4.1 Description de l'Algorithme

L'algorithme RandPEG proposé par Venkiah et al. [1], est une amélioration de l'algorithme PEG. Il permet de construire un code LDPC avec une girth cible en minimisant le nombre de cycles. Notre amélioration consiste essentiellement à ajouter une nouvelle contrainte dans la fonction objective du RandPEG. Cette contrainte ajoutée consiste à détecter les trapping set $(5,3)$, $(6,4)$, puis supprimer tous les noeuds de contrôle qui créent un trapping sets (5,3) et sélectionner seulement sur les noeuds de contrôle qui minimise le nombre de trapping sets $(6,4)$.

Considérons les codes LDPC dans $\mathcal{C}_{3,8}$. Nous cherchons à construire des codes LDPC sans trapping sets (5,3). Quand l'arbre de profondeur-k est étalé jusqu'à la profondeur $k_{max}\geq 3$, la *fonction coût* réduit l'ensemble des noeuds de contrôles candidats comme suit :

- Soit $\overline{\mathcal{M}}_v^{k_{max}}$, l'ensemble des noeuds de contrôle qui sont les candidats possibles pour la connexion de la nouvelle branche de la racine v de l'arbre de calcul. La première étape est de supprimer de $\overline{\mathcal{M}}_v^{k_{max}}$ tous les noeuds de contrôle qui apparaissent au moins une fois dans des profondeur inférieur à 3 dans l'arbre. Cette suppression permet d'éviter les cycles de taille inférieure à <8.
- Pour chacun des noeuds de contrôle c_m restant dans $\overline{\mathcal{M}}_v^{k_{max}}$, calcul $nbtrapping_m[i]_{0\leq i\leq 1}$, le nombre de trapping set (5,3) et (6,4) qui sont créés si c_m est sélectionné, en utilisant les théorèmes de détection 1 et 2. Supprime tous les noeuds de contrôle qui créent des trapping sets (5,3) $i.e$ noeud de contrôle c_m pour lesquels $nbtrapping_m[0]\neq 0$ et dans le but de minimiser le nombre de trapping sets (6,4), nous supprimons les noeuds de contrôle qui créent plus de $min_m(nbtrapping_m[1])$ trapping set (6,4).
- Si $\overline{\mathcal{M}}_v^{k_{max}}\neq\emptyset$, nous sélectionnons aléatoirement un noeud de contrôle c_m et nous connectons une nouvelle branche entre le noeud variable racine et c_m. Si $\overline{\mathcal{M}}_v^{k_{max}}=\emptyset$, un échec de conception est déclaré.

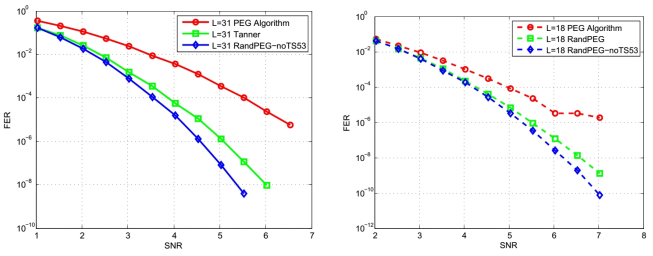
4.2 Résultats de Simulation

Nous nous focalisons sur la conception de code LDPC quasi-cyclique (QC-LDPC), qui sont les codes LDPC les plus utilisés dans les applications pratiques. La spécificité du PEG pour

	$L = 18, d_c = 5$			$L = 31, d_c = 5$			
	PEG	RandPEG	RandPEG no (5,3)	PEG	RandPEG	RandPEG no (5,3)	Tanner
$\#cycle - 6$	36	0	0	0	0	0	0
$\#cycle - 8$	684	774	720	682	620	527	465
$\#cycle - 10$	3402	3402	3582	3255	3348	3689	3720
$\#(5, 3; 8^3)$	0	144	0	62	31	0	155
$\#(6, 4; 8^2, 12)$	0	2286	1998	1271	930	434	0
$\#(6, 4; 8, 10^2)$	0	1530	1872	620	992	620	930

TABLE 1 – Comparaison des nombres de cycles et des nombres de trapping sets pour plusieurs constructions PEG

les QC-LDPC est d'abord l'expansion de l'arbre de calcul est seulement faite pour les noeuds variables $v = nL$, où L est la taille de la matrice de permutation circulante, et ensuite l'assignation de L branches dans une étape du PEG, par l'assignation entière d'un circulant. Dans le tableau 1, nous présentons les statistiques de deux codes LDPC réguliers avec $(d_v = 3, d_c = 5)$. Pour le cas $L = 31$, nous donnerons aussi les statistiques du code de Tanner proposé dans [8]. Pour l'algorithme PEG avec $L = 18$ la girth obtenue est $g = 6$ donc nous n'avons pas de $(5, 3)$ et $(6, 4)$, mais il est aussi démontré qu'avec des constructions PEG-like plus efficace, une girth $g = 8$ est atteinte pour ce rendement et cette longueur. Les statistiques dans ce tableau montrent clairement que notre nouvelle contrainte dans la conception évite tous les TS (5,3), et aussi minimise le nombre de TS(6,4), comparé à la construction RandPEG original. Fig.3(a) et Fig.3(b) montrent les performances, sur un canal BPSK-AWGN, des codes LDPC réguliers listé dans le tableau 1. Nous pouvons voir que l'algorithme PEG peut être grandement amélioré en utilisant l'algorithme RandPEG, et plus encore avec notre amélioration. Noter que pour ces mots codes de longueur faible, nous n'avons pas atteint la région du plancher d'erreur dans nos simulations (excepté pour le PEG, $L = 18$ code qui a une girth 6), mais l'amélioration de la pente est significative et montre que nous avons supprimé beaucoup de point immobile dominant du décodeur itératif.



(a) (b)

FIGURE 3 – Comparaison des performance des QC-LDPC avec $d_v = 3$, $d_c = 5$, et L=18 pour (b), L=31 pour (a)

5 Conclusions

Dans ce papier, nous avons proposé une méthode prédictive pour construire les codes LDPC de poids colonne régulier avec une girth $g = 8$ tel que leurs graphes de Tanner n'aient pas de trapping set $(5, 3)$ et un nombre minimum de trapping sets $(6, 4)$. Notre construction est basé sur l'algorithme Randomized Progressive Edge Growth (RandPEG). Les statistiques sur les cycles et les faibles TS ainsi que les simulations Monte Carlo démontrent l'avantage de notre approche pour les codes QC-LDPC de longueurs faibles, comparé aux modèles existants.

Remerciements

B. Vasic tient à souligner l'appui de la NSF en vertu de concessions CCF-0963726 et 1314147-CCF, et le Programme de bourse d'études Fulbright.

Références

[1] A. Venkiah, D. Declercq, and C. Poulliat, "Design of Cages with a Randomized Progressive Edge Growth Algorithm", *IEEE Commun. Letters*, vol. 12, pp. 301-303, April 2008.

[2] T. J. Richardson, "Error Floors of LDPC Codes", *Proc. 41st Annual Allerton Conf. on Communications, Control and Computing*, pp. 1426-1435, 2003.

[3] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and Irregular Progressif Edge-Growth Tanner Graphs", *IEEE Trans. Inf. Theory*, vol. 51, no. 01,pp. 386-398, Jan. 2005.

[4] , G. Richter, et A. Hof, "On a Construction Method of Irregular LDPC Codes Without Small Stopping Sets", *Proc. IEEE Int. Conf.* , Istanbul, Turkey, vol. 3, pp. 1119-1124, June 2006.

[5] V. Vasic, S. K. Chilappagari, D. V. Nguyen, et S. K. Planjery,"Trapping set ontology", *Proc. 47th Annual Allerton Conf. on Commun., Control, and Computing* , pp. 1-7, Sept. 2009,

[6] , M. Karimi et A. Banihashemi, "On Characterization of Elementary Trapping Sets of Variable-Regular LDPC Codes"', *IEEE Trans. Inf. Theory*, vol. 60, no. 09, pp. 5188–5203, Sept. 2014.

[7] , D. Declercq, B. Vasic, S. K. Planjery, et E. Li, "Finite Alphabet Iterative Decoders, Part II : Improved Guaranteed Error Correction of LDPC Codes via Iterative Decoder Diversity", *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4046-4057, Nov. 2013.

[8] , M. Tanner, D. Sridhara, et T. Fuja, "A class of group-structured LDPC codes", [Online] citeseer.ist.psu.edu/tanner01class.html, 2001.