Université de Montréal

Générateurs de nombres pseudo-aléatoires utilisant des récurrences linéaires modulo 2

par
François Panneton
Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique option recherche opérationnelle

Septembre 2000

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Générateurs de nombres pseudo-aléatoires utilisant des récurrences linéaires modulo 2

présenté par:

François Panneton

a été évalué par un jury composé des personnes suivantes:

Patrice Marcotte

(président-rapporteur)

Pierre L'Écuyer

(directeur de recherche)

Gilles Brassard

(membre du jury)

Mémoire accepté le:

Sommaire

Ce mémoire porte sur les générateurs de nombres pseudo-aléatoires qui utilisent une récurrence linéaire avec l'arithmétique modulo 2. Ces générateurs sont rapides, car ils tirent avantage de l'architecture des ordinateurs binaires où l'information est emmagasinée avec seulement deux symboles : 0 et 1. De plus, la structure linéaire de ces générateurs permet d'étudier certains critères d'uniformité des valeurs générées. En vérifiant ces critères, il est possible d'obtenir de bons générateurs pour la simulation Monte Carlo et l'intégration numérique utilisant des méthodes de type quasi-Monte Carlo.

Ce mémoire présente un aperçu des principaux générateurs à récurrence linéaire modulo 2 ainsi que la manière de vérifier leur équidistribution, principal critère abordé dans ce mémoire, qui est une mesure de l'uniformité des points générés. On aborde aussi des méthodes utilisées afin d'améliorer l'équidistribution.

Le travail le plus important de ce mémoire est le développement du progiciel REGPOLY. Ce progiciel permet de vérifier l'équidistribution des générateurs à récurrences linéaires modulo 2. Un programme, utilisant le progiciel, a été développé afin de tirer le maximum des capacités de REGPOLY. Ce programme a été utilisé afin de trouver des générateurs de type GCL polynomial et des générateurs de type TGFSR à deux ou trois composantes qui démontrent des équidistributions satisfaisantes. Une liste de ces générateurs est présentée.

Table des matières

Liste des figures

Liste des tableaux

Liste des sigles et abréviations

$\mathbf{a}$	Vecteur de bits $\mathbf{a}$.
$a^{(i)}$	Le bit numéro i de $\mathbf{a}$. Les bits sont numérotés de gauche vers la droite en partant de 0.
k	Degré du polynôme caractéristique du générateur.
S	Espace des états du générateur.
U	Espace des sorties du générateur.
$\mathbf{x}_n$	État du générateur à la n-ième itération.
$\mathbf{z}_n$	Résultat de la muliplication de $\mathbf{x}_n$ par B à la n-ième itération.
$\mathbf{y}_n$	Vecteur de sortie du générateur à la n-ième itération.
u_n	Sortie du générateur à la n-ième itération comprise dans l'intervalle $(0, 1]$.
X	Matrice de la récurrence du générateur.
Y	Matrice qui multiplie $\mathbf{z}_n$ pour obtenir $\mathbf{y}_n$.
B	Matrice représentant une transformation linéaire appliquée au vecteur d'état. Multiplie $\mathbf{x}_n$ pour obtenir $\mathbf{z}_n$.
H	Matrice résultant de la multiplication des matrices B et Y
w	Nombres de bits de $\mathbf{x}_n$ qui seront utilisés pour la transformation linéaire .
L	Résolution de la sortie du générateur .
J	Nombre de composantes pour le générateur combiné

j	Numéro d'une composante d'un générateur combiné
k'	$\min(k, L)$
k'_j	$\min(k_j, L_j)$
$T_{w,s,t,\mathbf{b},\mathbf{c}}$	Tempering de M-K avec paramètres $s, t, \mathbf{b}$ et $\mathbf{c}$.
$S_{c,d}$	Self-tempering de paramètres c et d.
P	Permutation de coordonnées.
$P_{p,q}$	Permutation de coordonnées avec $\pi(i) = pi + q \bmod k$.
$\tilde{P}^w_{p,q}$	Permutation de coordonnées avec $\pi(i) = pi + q \bmod w$
$\mathbf{a}^T$	Transposée du vecteur $\mathbf{a}$.
B^T	Transposée de la matrice B.
B^{-1}	Inverse de la matrice B.
$\mathrm{pgcd}(a, b)$	Plus grand commun diviseur de a et b.
$\mathrm{ppcm}(a, b)$	Plus petit commun multiple de a et b.
$\mathbb{F}_2$	Corps fini à deux éléments.
$\mathbb{F}_{2^k}$	Corps fini à 2^k éléments.
$\mathbb{F}_2[z]$	Ensemble des polynômes ayant des coefficients dans $\mathbb{F}_2$.
$\mathbb{Z}_k$	L'ensemble$\{0, 1, \ldots, k-1\}$.
$\deg(p)$	Degré du polynôme p.
I_k	Matrice identité $k \times k$ dans $\mathbb{F}_2$.
$I_{L\times k}$	Matrice $L \times k$ ne contenant que les L premières lignes de I_k.
$\mathbf{a} \oplus \mathbf{b}$	Ou-exclusif bit par bit entre $\mathbf{a}$ et $\mathbf{b}$.
$\mathbf{a} \ll j$	Décalage de j bits vers la gauche du vecteur $\mathbf{a}$.
$\mathbf{a} \gg j$	Décalage de j bits vers la droite du vecteur $\mathbf{a}$.
$\mathbf{a} \tilde{\ll} j$	Décalage rotatif de j bits vers la gauche du vecteur $\mathbf{a}$.
$\mathbf{a} \tilde{\gg} j$	Décalage rotatif de j bits vers la droite du vecteur $\mathbf{a}$.
$\lfloor x \rfloor$	Plus grand entier inférieur ou égal à x.

$\lceil x \rceil$	Plus petit entier supérieur ou égal à x.
$\mathrm{trunc}_w(\mathbf{x})$	Opération qui retourne un vecteur de w bits contenant les w premiers bits de $\mathbf{x}$
Φ_1	Ensemble contenant les dimensions $t \leq \sqrt{k}$ pour lesquelles il faut vérifier l'équidistribution
Φ_2	Ensemble contenant les dimensions $t > \sqrt{k}$ pour lesquelles il faut vérifier l'équidistribution
Ψ_1	Ensemble contenant les résolutions $l \leq \sqrt{k}$ pour lesquelles il faut vérifier l'équidistribution
Ψ_2	Ensemble contenant les résolutions $l > \sqrt{k}$ pour lesquelles il faut vérifier l'équidistribution
GFSR	General Feedback Shift Register
TGFSR	Twisted General Feedback Shift Register
GCL	Générateur à congruence linéaire
MT	Mersenne Twister

Remerciements

Je remercie d'abord Pierre L'Écuyer, mon directeur de travaux, pour toute l'aide qu'il m'a apportée tout au long de ce travail de maîtrise ainsi que pour m'avoir initié à la simulation par ordinateur par l'entremise de l'étude des générateurs de nombres aléatoires.

Je remercie également Raymond Couture, pour avoir eu l'idée de faire des permutations sur l'état des générateurs et de faire évoluer les récurrences dans l'état transformé des générateurs dans ce cas particulier.

Je remercie aussi le CRSNG pour son support financier tout au long de ma maîtrise.

Ma copine, Mélanie Montreuil, a été d'un très grand support moral et ses encouragements continuels m'ont permis de persévérer. Je la remercie sincèrement.

Je souligne le soin que mes parents, Johanne et Michel, ont pris afin que j'obtienne une bonne éducation sans laquelle la réalisation de ce mémoire n'aurait été possible.

Je remercie également Renée Touzin et Jacinthe Granger-Piché pour avoir corrigé mes textes.

Chapitre 1

Introduction

1.1 Comment imiter le hasard

Lors de la simulation par ordinateur, il est essentiel de pouvoir imiter le hasard. Cette tâche n'est pas facile pour un ordinateur. Il s'agit de produire une suite de valeurs qui imite une suite de variables aléatoires uniformes indépendantes dans l'intervalle [0,1). Les principales propriétés désirées qui permettent de choisir les méthodes à utiliser sont expliquées dans [14]. Voici un résumé de celles-ci :

1. bonnes propriétés statistiques : on désire obtenir une séquence de nombre $u_0, u_1, \ldots$ qui passe avec succès la plupart des tests statistiques raisonnables.
2. longue période : supposons que l'on ait besoin de N valeurs aléatoires pour une simulation, alors il faut que la période (le nombre de valeurs produites avant que la séquence ne se répète) des valeurs produites soit beaucoup plus grande que N ;
3. efficacité : le temps de calcul nécessaire afin de produire les valeurs doit être négligeable par rapport au temps de la simulation. Ce facteur devient important dans des simulations qui prennent plusieurs jours d'exécution ;
4. répétabilité : l'utilisateur doit être capable de reproduire la même séquence de nombres facilement. Ceci est important pour la vérification des programmes et certaines techniques de réduction de variance.

5. facilité d'implantation et séparabilité : le générateur doit pouvoir être exécuté sur tous les types d'ordinateurs standards. Également, il doit être facile de "sauter" d'une valeur u_n à une autre u_{n+s} facilement, même quand s est grand. De cette manière, on peut utiliser plusieurs sous-séquences de la séquence $u_0, u_1, ..$ et considérer chaque sous-séquence comme un générateur indépendant (ceci exige que la période de la séquence $u_0, u_1, ..$ soit assez grande pour le permettre).

Au critère 1, un test statistique raisonnable est un test qui peut être effectué dans un temps raisonnable. Par exemple, s'il faut 200 ans avant de trouver un défaut sur un générateur à l'aide d'un test statistique, alors on pourrait dire que le test n'est pas raisonnable. Ces tests sont standardisés et peuvent être trouvés dans [11, 5, 9, 24].

Il existe deux types de méthode permettant de reproduire le hasard et qui tentent de répondre à ces critères. Il y a les méthodes physiques et les méthodes mathématiques.

Les premières méthodes, les méthodes physiques, produisent des valeurs "vraiment aléatoires". Par exemple, on pourrait utiliser une pièce de monnaie et enregistrer les résultats des tirages successifs de celle-ci. On obtiendrait alors une suite binaire aléatoire. D'autres méthodes peuvent être utilisées et les résultats observés seraient aléatoires dans le sens que ceux-ci suivent une distribution de probabilité qui est propre à l'expérience choisie.

Les méthodes physiques ont plusieurs désavantages qui les rendent inintéressantes pour la simulation. Le critères 3 est difficilement respecté puisque le nombre de valeurs disponibles est limité par le temps nécessaire pour produire ces valeurs et/ou l'espace de stockage de l'information. Si le nombre de valeurs produites est trop faible, alors celles-ci auront vite été utilisées lors d'une simulation. Le critère 1 n'est pas facilement respecté étant donné la difficulté d'obtenir des valeurs successives vraiment indépendantes les unes des autres et aussi le fait qu'il est difficile de s'assurer que les valeurs produites suivent une distribution uniforme dans l'intervalle [0,1). Ces méthodes sont rarement utilisées de nos jours. Les références [2] et [10] traitent plus en détail des méthodes physiques.

Le deuxième type de méthodes, les méthodes mathématiques, permet de produire une suite de nombres qui ressemble à celle que produirait un système parfaitement aléatoire. Les nombres produits sont alors appelés *nombres pseudo-aléatoires* car ils ne sont pas générés par une méthode aléatoire, mais plutôt par une méthode déterministe.

Les méthodes mathématiques ont l'avantage de ne pas nécessiter d'espace de stockage permanent et de produire des valeurs pseudo-aléatoires rapidement, ce qui répond au critère 3. Aussi, le critère 2 est respecté puisque les méthodes actuelles ont de longues périodes, c'est-à-dire, que les simulations utilisent rarement toutes les valeurs disponibles. C'est à cause de leur rapidité et leur facilité d'utilisation que l'on utilise des méthodes mathématiques pour la majorité des simulations. Le modèle mathématique qui permet de produire la séquence de nombres aléatoires est appelé *générateur de nombres pseudo-aléatoires*. Une propriété des méthodes mathématiques est qu'elles ont toujours une période finie. Afin d'alléger le texte, on parlera plutôt d'un *générateur de nombres aléatoires* ou *générateur*.

Idéalement, au lieu du critère 1, on voudrait que le générateur soit imprévisible, c'est-à-dire que l'on voudrait que l'on ne puisse faire la différence entre une séquence produite par un générateur donné et une autre suite de variables aléatoires indépendantes identiquement distribuées uniformes dans l'intervalle [0,1) avec une probabilité significativement supérieure à 1/2 [12]. Évidemment, ceci n'est pas possible, puisqu'en examinant la séquence des nombres générés assez longtemps, il est possible de déterminer de façon exacte la prochaine valeur du générateur, puisque la séquence est périodique. Pour certains générateurs, il est même possible de connaître tous les paramètres du modèle, même avec une petite fraction de la période [7, 29]. Quelques générateurs sont presque imprévisibles [21, 1], mais ceux-ci sont trop lents pour les besoins pratiques de la simulation [12].

Un problème des générateurs de nombres aléatoires est que les valeurs produites ne sont pas aléatoires, ils ne font qu'imiter des variables aléatoires. Ceci engendre des structures dans les ensembles de points générés. C'est pour cette raison qu'il faut être très prudent lorsque l'on utilise une méthode mathématique. Dans le passé, des gens ont conçu des générateurs sans en étudier à fond leurs propriétés. Un générateur célèbre pour ses lacunes est le générateur RANDU qui a été implanté dans les ordinateurs d'IBM dans les années 1960. Voir [6] et [8] pour plus de détails sur ce générateur et ses défaillances.

1.2 Définition du générateur de nombres aléatoires

Il est maintenant temps de donner une définition d'un générateur de nombres aléatoires. Pour le besoin de ce mémoire, nous utilisons la définition qui est donnée dans [12]. Tout d'abord, définissons un espace fini d'états S et choisissons un élément de S, soit s_0, qui est l'état initial de l'algorithme appelé *racine* (qu'on appelle aussi *germe*) du générateur. Construisons une fonction $f : S \to S$, appelée *fonction de transition*, qui permet de passer d'un état à l'autre à l'aide de la récurrence $s_n = f(s_{n-1})$. À chaque état s_n correspond un nombre u_n compris dans un ensemble U. Afin de trouver ce nombre u_n, construisons une fonction $g : S \to U$ qu'on appelle *fonction de sortie*. Dans la majorité des cas, $U = [0, 1)$.

Le générateur produit, à chacune des itérations, un nouvel état $s_n \in S$ avec f et une sortie $u_n \in U$, obtenue avec g. La structure est notée $G = (S, s_0, f, U, g)$. On peut aussi facilement remarquer que, puisque le nombre d'états est fini, la séquence des nombres en sortie (les u_n) est périodique. La période maximale d'un générateur est $|S|$. La construction déterministe des générateurs de nombres aléatoires est quelque peu contradictoire, mais du point de vue pratique c'est une approche qui fonctionne bien si les générateurs ont de bonnes propriétés.

1.3 Bref aperçu des générateurs utilisés dans ce mémoire

Dans ce mémoire, il est question de générateurs de nombres pseudo-aléatoires qui utilisent une récurrence linéaire modulo 2. L'espace des états est $\mathbb{F}_2^k$, un ensemble contenant 2^k éléments. Un élément de cette ensemble peut être représenté par un vecteur de k bits, puisqu'il existe 2^k vecteurs de ce type. La représentation par vecteurs de bits est utilisée afin de passer d'un état à l'autre, puisque celle-ci est facilement manipulée par les ordinateurs actuels. Les générateurs de ce type déjà existants sont les générateurs de Tausworthe [30, 13, 17], les TGFSR [25, 26], les GCL polynomiaux [20] et les Mersenne Twister [27]. Tous ces générateurs utilisent, sous des formes différentes, le même type de récurrence linéaire et peuvent tous être représentés sous une forme matricielle.

Si on représente l'état courant par un vecteur colonne de k bits

$$\mathbf{x}_{n-1} = (x_{n-1}^{(0)}, \ldots, x_{n-1}^{(k-1)})^T,$$

alors le prochain état peut être obtenu par l'équation

$$\mathbf{x}_n = X\mathbf{x}_{n-1} \tag{1.1}$$

où X est une matrice inversible $k \times k$ dont chaque élément est élément de $\mathbb{F}_2$. Les opérations inhérentes à la multiplication par une matrice (multiplications et additions) se font "modulo 2", ou de façon équivalente, "dans $\mathbb{F}_2$". Tout au long de ce mémoire, nous utilisons cette arithmétique pour la manipulation des vecteurs de bits et de matrices de bits. La matrice X correspond donc à la fonction f de la définition du générateur de nombres aléatoires. La fonction g, qui permet d'obtenir la sortie est donnée par les équations

$$\mathbf{y}_n = H\mathbf{x}_n \tag{1.2}$$

et

$$u_n = \sum_{i=1}^{L} 2^{-i} y_n^{(i-1)} \tag{1.3}$$

où $\mathbf{y}_n = (y_n^{(0)}, \ldots, y_n^{(L-1)})^T$ est un vecteur de L bits, H est une matrice $L \times k$ avec ses entrées dans $\mathbb{F}_2$ et L est la résolution (le nombre de bits) que l'on désire pour la sortie u_n comprise dans l'intervalle [0,1). On obtient alors que $g(\mathbf{x}_n) = u_n$. Plus loin dans ce mémoire, la matrice H sera décomposée de telle manière que $H = YB$. Les raisons de cette décomposition sont multiples et seront expliquées aux chapitres 2, 4 et 5.

En observant les équations (1.1), (1.2) et (1.3), on remarque que l'on peut obtenir différentes suites de nombres $u_0, u_1, \ldots$ avec la même matrice X en utilisant différentes matrices H. Cette observation se révèle importante lorsqu'on désire améliorer ou changer les propriétés des générateurs. Les conditions sur la matrice X sont plus fortes que celles imposées à la matrice H, ce qui rend la matrice H plus facile à changer que la matrice X.

En changeant la matrice H, on peut obtenir différents générateurs avec la même matrice X. Pour une matrice X, on sélectionne les matrices H selon un critère qui mesure la qualité du générateur obtenu. Le critère qui fait office de mesure de la qualité des générateurs de nombres aléatoires à récurrences linéaires modulo 2 est l'équidistribution que nous définissons dans la prochaine section.

1.4 Bref aperçu du critère d'équidistribution

Pour les générateurs utilisant une récurrence linéaire modulo 2, l'équidistribution est la méthode privilégiée afin de vérifier leur qualité. L'équidistribution, que nous définissons dans cette section, est une heuristique qui sert à mesurer l'uniformité des points produits par le générateur dans des hypercubes en t dimensions pour plusieurs valeurs de t. Cette méthode se classe parmi celles qui vérifient les propriétés des générateurs de manière théorique.

Comme dans [13, 14], définissons Ω_t, l'ensemble de tous les vecteurs de t valeurs successives produites par le générateur, à partir de toutes les 2^k racines possibles. C'est-à-dire

$$\Omega_t = \{\mathbf{u}_{0,t} = (u_0, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}.$$

Pour une valeur de ℓ donnée, on partitionne chaque axe de l'hypercube $[0,1)^t$ en 2^ℓ parties égales. Ceci détermine une partition de l'hypercube en $2^{\ell t}$ petits cubes de volumes égaux. Le générateur est dit *(t,ℓ)-équidistribué* si chaque petit cube contient exactement $2^{k-\ell t}$ points de Ω_t. Ceci revient à dire que si l'on considère les ℓ bits les plus significatifs de chacune des coordonnées de $\mathbf{u}_{0,t}$, alors chacun des $2^{\ell t}$ vecteurs possibles apparaît exactement le même nombre de fois dans Ω_t. Ceci n'est possible que si $\ell t \leq k$. Si le générateur est $(\lfloor k/\ell \rfloor, \ell)$-équidistribué pour $1 \leq \ell \leq \min(k, L)$, alors le générateur est dit *équidistribué au maximum* ou ME (de l'anglais *maximally equidistributed*).

Dans la littérature, lorsque vient le temps de vérifier la qualité des générateurs de nombres aléatoires, il existe deux types de méthodes : les méthodes empiriques et les méthodes théoriques. Dans [15], on conclut qu'il est préférable de considérer les propriétés théoriques en premier lieu et ensuite, si les résultats sont satisfaisants, de vérifier les propriétés empiriquement. Aussi, on montre que si un générateur satisfait à un critère théorique exigeant, les chances d'échouer des critères empiriques diminuent considérablement. Par contre, étant donné le caractère déterministe des générateurs de nombres aléatoires, il est toujours possible de faire échouer un générateur à un test statistique. Dans [18], on propose certaines façons d'utiliser les générateurs afin que ces défauts statistiques n'affectent pas la simulation.

Également dans [18], on affirme que si Ω_t couvre bien l'hypercube $[0,1]^t$ pour plusieurs valeurs de t et si le germe est choisi au hasard les $\mathbf{u}_{0,t}$ suivent une distribution qui se rapproche de celles de variables aléatoires uniformes et indépendantes les unes des autres. Ainsi l'équidistribution est une heuristique visant à obtenir des valeurs successives indépendantes et uniformes.

1.5 Générateurs combinés

Il est possible de combiner plusieurs générateurs. La manière de combiner les générateurs utilisant des récurrences linéaires modulo 2 est simple. Il s'agit de faire fonctionner les générateurs en parallèle et, à chaque itération, on produit une sortie en combinant toutes les sorties de chacun des générateurs (par exemple, addition modulo 1, ou-exclusif bit à bit).

Soit J générateurs distincts. Pour le type de combinaisons qui vont nous intéresser, la période du générateur combiné est donnée par (voir [13])

$$\mathrm{ppcm}(\rho_1, \rho_2, \ldots, \rho_J) \tag{1.4}$$

où ρ_j est la période de la composante j. Il est évident, qu'afin d'avoir la période maximale pour le générateur combiné, il est nécessaire que $\mathrm{pgcd}(\rho_1, \rho_2, \ldots, \rho_J)$=1. Au chapitre 2, on décrit plus en détails les combinaisons que l'on utilisera.

Afin de bien définir les générateurs combinés, il faut introduire une certaine notation. Alors, définissons, à la n-ième itération, l'état de la j-ième composante $\mathbf{x}_{n,j} = (x_{n,j}^{(0)}, \ldots, x_{n,j}^{(k_j-1)})$ où k_j est le nombre de bits sur lesquels se définit l'état de la composante. Le vecteur de sortie de la j-ième composante est $\mathbf{y}_{n,j} = (y_{n,j}^{(0)}, \ldots, y_{n,j}^{(L_j-1)})$ où L_j est la résolution de la composante. Pour chaque composante j, il faut également définir H_j, une matrice $L_j \times k_j$ de plein rang. On définit ensuite pour chacune des composantes

$$\mathbf{y}_{n,j} = H_j \mathbf{x}_{n,j}. \tag{1.5}$$

Il existe plusieurs façons de combiner les sorties de chacune des composantes, mais pour les besoins de ce mémoire, on obtient le vecteur de sortie du générateur combiné $\mathbf{y}_n$ par

$$\mathbf{y}_n = \mathbf{y}_{n,1} \oplus \ldots \oplus \mathbf{y}_{n,J} \tag{1.6}$$

où l'opération $\oplus$ signifie une addition modulo 2 bit par bit, la valeur de L est définie par

$$L = \min \{L_j \mid 1 \leq j \leq J\} \tag{1.7}$$

et la sortie u_n est déterminée par la formule habituelle (1.3).

1.6 Les transformations linéaires

Dans ce mémoire, nous essayons de tirer avantage de la liberté offerte par la matrice H, afin d'améliorer la qualité de l'équidistribution. Dans [13, 17, 30, 32, 25], si $L \leq k$, on utilise de façon implicite $H = I_{L \times k}$ qui est est une matrice $L \times k$ ne contenant que les L premières lignes de la matrice identité I_k de dimension $k \times k$. Dans le cas où $L > k$, on utilise la matrice $H = (I_k\ C)^T$, où C est une matrice $(L-k) \times k$. Autrement dit, quelque soit le choix de H, les $\min(k, L)$ premiers bits de $\mathbf{y}_n$ et $\mathbf{x}_n$ étaient toujours les mêmes.

L'article [26] décrit la première utilisation d'une transformation linéaire H autre que l'identité. Cette transformation linéaire est le tempering de Matsumoto-Kurita. Dans cet article, on présente un algorithme qui permet de choisir adéquatement les paramètres du tempering de Matsumoto-Kurita afin d'obtenir la meilleure équidistribution possible pour un TGFSR. Dans ce mémoire, nous introduisons un algorithme plus général qui permet d'obtenir la meilleure équidistribution possible pour des générateurs combinés où chaque composante a un tempering de Matsumoto-Kurita qui lui est propre. Cet algorithme permet au progiciel REGPOLY, que nous allons décrire plus loin dans ce mémoire, de trouver des générateurs TGFSR à deux ou trois composantes qui sont très bien équidistribués, mieux que dans [26].

Nous introduisons également deux nouvelles transformations linéaires qui permettent d'améliorer l'équidistribution. Ces deux transformations sont appelées *permutation* et *self-tempering*. Ces transformations linéaires ont été utilisées, afin de permettre au progiciel REGPOLY de trouver des générateurs de type GCL polynomial qui sont ME.

1.7 Le progiciel REGPOLY

Afin de trouver des générateurs à récurrence linéaire modulo 2 qui démontrent une bonne équidistribution (ou qui excellent par rapport à d'autres critères), nous avons développé le progiciel REGPOLY. Ce progiciel est inspiré de deux programmes qui, pour le premier, vérifiait l'équidistribution de générateurs de Tausworthe combinés et, pour le deuxième, (qui était d'ailleurs incomplet) vérifiait l'équidistribution de TGFSR combinés sur lesquels on utilisait un tempering de Matsumoto-Kurita sur chacune des composantes. Ces programmes n'étaient pas pratiques puisque leurs codes, quoique similaires, n'étaient nullement compatibles. De plus, ces programmes permettaient peu de flexibilité par rapport aux matrices H à essayer. Pour le premier programme, on ne permettait que $H = (I_k\ C)^T$ et, pour le second, seule la matrice H du tempering de Matsumoto-Kurita était permise.

Le progiciel REGPOLY permet de vérifier l'équidistribution (ou d'autres critères) sur n'importe quel générateur qui entre dans le cadre défini par les équations (1.1), (1.2) et (1.3). Le progiciel est en fait un ensemble d'outils qui permet à un programmeur de fabriquer ses propres programmes de recherche de générateurs à récurrence linéaire modulo 2. La modularité du progiciel laisse la porte ouverte à des ajouts de modules futurs.

En fait, le progiciel REGPOLY, dont le guide est disponible en annexe de ce mémoire, est la principale contribution de ce travail de maîtrise. Beaucoup de temps a été consacré afin de généraliser le plus possible le problème de la recherche de générateurs de nombres aléatoires combinés qui utilisent une récurrence linéaire modulo 2.

1.8 Aperçu du mémoire

Dans le prochain chapitre, il est question de l'aspect théorique de la récurrence utilisée pour les générateurs de nombres aléatoires. Les équations (1.1), (1.2) et (1.3) sont expliquées en détail et nous démontrons comment chacun des types de générateurs traités dans ce mémoire (GCL polynomial, Tausworthe, TGFSR et Mersenne twister) entre dans ce cadre. On discute également de la manière de combiner ces générateurs pour n'en former qu'un seul.

Dans le chapitre 3, nous élaborons sur la manière de vérifier l'équidistribution des générateurs à récurrence linéaire modulo 2. Il y a également une section qui traite de l'équidistribution des projections dans des dimensions à indices non successifs.

Le chapitre 4 explique en détail trois transformations linéaires H qui permettent d'obtenir des générateurs ayant de bonnes équidistributions. Ces transformations sont la permutation de coordonnées, le self-tempering et le tempering de Matsumoto-Kurita. On décrit également l'algorithme qui permet de trouver de bons paramètres pour le tempering de Matsumoto-Kurita.

Les algorithmes qui permettent d'implanter les différents générateurs et les différentes transformations linéaires forment le sujet du chapitre 5.

Dans le chapitre 6, on traite du développement du progiciel REGPOLY et de ses possibilités. Le chapitre 7 présente les résultats des recherches effectuées par le programme `recherche.c` écrit avec le progiciel REGPOLY. Les générateurs recherchés sont des générateurs de type GCL polynomial à une seule composante et les TGFSR à deux ou trois composantes. Également, en annexe de ce mémoire, on présente le guide d'utilisation du progiciel REGPOLY.

Chapitre 2

Théorie des générateurs utilisant une récurrence linéaire modulo 2

Dans ce chapitre, il est question de la théorie des générateurs utilisant des récurrences linéaires modulo 2. On discute des générateurs de types GCL polynomial, Tausworthe, TGFSR et Mersenne twister. Il est à remarquer que tous ces types de générateurs forment une façon différente de représenter la même récurrence.

2.1 Bref aperçu des corps finis

Beaucoup de générateurs peuvent être vus comme une récurrence linéaire dans un corps fini S. Le livre de Lidl et Niederreiter [23] couvre la majorité des aspects mathématiques des corps finis discutés dans ce mémoire. La définition suivante permet de mieux comprendre la notion de corps fini.

Définition 2.1.1 *Groupe[28]*
Un groupe est un ensemble G sur lequel est défini une opération "+" qui est associative. De plus, il y a un élément "0", appelé "élément neutre", dans G tel que $0 + g = g + 0$ pour tout $g \in G$ et chaque élément $g \in G$ a un inverse $h \in G$ tel que $h + g = g + h = 0$.

Définition 2.1.2 *Groupe Abélien[28]*

Un groupe G est dit Abélien *si l'opération "·" est commutative.*

Définition 2.1.3 *Corps[28].*

Un Corps est un ensemble Q comprenant 2 éléments, "0" et "1", combiné avec 2 opérations, "+" et "·", tel que

1. *Q est un groupe abélien sous l'opération "+", avec l'élément neutre "0".*
2. *Les éléments autres que "0" de Q forment un groupe abélien (où "1" est l'élément neutre) sous l'opération "·".*
3. *La propriété $a \cdot (b + c) = a \cdot b + a \cdot c$ est valide.*

Un corps est appelé corps fini, si Q est un ensemble fini.

Exemple

Les tableaux 2.1 et 2.2 montrent les tables d'opération pour "+" et "·" dans $\mathbb{F}_2$. Le corps pris en exemple est $\mathbb{F}_2$, puisque ce corps est celui le plus utilisé dans ce mémoire. Pour $\mathbb{F}_2$, on a $Q = \{0, 1\}$

Tableau 2.1 – Table de l'addition dans $\mathbb{F}_2$

+	0	1
0	0	1
1	1	0

Tableau 2.2 – Table de la multiplication dans $\mathbb{F}_2$

.	0	1
0	0	0
1	0	1

Dans le cas qui nous intéresse, $S = \mathbb{F}_2^k$, l'ensemble des vecteurs de k bits. L'ensemble $\mathbb{F}_2^k$ a donc 2^k éléments et si $k = 1$, alors $S = \mathbb{F}_2$ est le corps fini à 2 éléments. On peut également avoir $S = \mathbb{F}_{2^k}$, comme dans le cas du GCL polynomial qui sera défini plus loin dans ce chapitre. Dans ce cas, S est un corps fini à 2^k éléments. On peut construire le corps fini $\mathbb{F}_{2^k}$ de plusieurs façons. Mais peu importe la manière, il existe toujours un isomorphisme entre celles-ci. Soit $\mathbb{F}_2[z]$, l'ensemble de tous les polynômes à coefficients éléments de $\mathbb{F}_2$. Si $k \geq 1$, on peut construire $\mathbb{F}_{2^k}$ avec l'anneau de polynômes $\mathbb{F}_2[z]/(P(z))$, qui est l'espace des polynômes modulo $P(z)$ avec coefficients dans $\mathbb{F}_2$ ($P(z)$ est un polynôme irréductible de degré k avec coefficients dans $\mathbb{F}_2$ [23]).

Voici maintenant quelques définitions qui permettent de mieux comprendre le reste de ce chapitre.

Définition 2.1.4 *Polynôme irréductible.*
Un polynôme, ayant comme coefficients des éléments de $\mathbb{F}_2$, est irréductible, *s'il ne peut être factorisé en des polynômes non triviaux. Par exemple, x^2+x+1 est irréductible, mais x^2+1 ne l'est pas puisque $x^2+1 = (x+1)(x+1)$.*

Définition 2.1.5 *Ordre d'un polynôme.*
L'ordre d'un polynôme $f(x)$ pour lequel $f(0) \neq 0$ est le plus petit entier e tel que $f(x)$ divise x^e+1.

Définition 2.1.6 *Polynôme primitif dans $\mathbb{F}_2[z]$.*
Un polynôme de degré k est dit primitif *dans $\mathbb{F}_2[z]$ s'il est d'ordre 2^k-1. Par exemple, x^2+x+1 est d'ordre $3 = 2^2-1$ puisque $(x^2+x+1)(x+1) = x^3+1$, $(x^2+x+1) \not| (x^2+1)$ et $(x^2+x+1) \not| (x+1)$. C'est pourquoi x^2+x+1 est primitif.*

Définition 2.1.7 *Générateur du groupe cyclique.*
L'élément α du groupe cyclique Q est appelé générateur *si tous les éléments du groupe peuvent être exprimés par α^s, où $0 \leq s \leq |Q|$.*

2.2 Récurrence de base pour les générateurs utilisant une récurrence linéaire modulo 2

Pour les générateurs basés sur une récurrence linéaire modulo 2, la récurrence de base est

$$x_j = (a_1 x_{j-1} + ... + a_k x_{j-k}) \bmod 2, \tag{2.1}$$

où k est appelé l'ordre de la récurrence si $a_k \neq 0$ et $a_i \in \mathbb{F}_2$ pour $i = 1, \ldots, k$. À cette récurrence est associée un polynôme que l'on appelle *polynôme caractéristique de la récurrence*. Ce polynôme est

$$P(z) = z^k - a_1 z^{k-1} - \cdots - a_k.$$

Grâce à ce polynôme, on peut déduire des informations importantes sur la récurrence. Une de ces informations est la période de la récurrence. En effet, il est bien connu que le générateur sera de période maximale $2^k - 1$ si et seulement si le polynôme caractéristique est primitif [23]. Pour tous les générateurs de ce mémoire, il est possible de montrer qu'ils utilisent une récurrence de la forme (2.1).

2.3 Générateurs utilisant une récurrence linéaire modulo 2 : cadre général

Avant de décrire spécifiquement chacun des générateurs utilisés dans ce mémoire, il est important de décrire le cadre général utilisé par chacun des générateurs. En partant de la définition des générateurs de nombres aléatoires introduite à la section 1.2, on définit l'espace d'états S, la fonction de transition f et la fonction de sortie g utilisés pour les générateurs à récurrence linéaire modulo 2.

Pour les générateurs utilisés, l'espace d'états S est $\mathbb{F}_2^k$. Cet espace d'états est représenté par un vecteur de k bits. Le n-ième état sera noté $\mathbf{x}_n = (x_n^{(0)}, x_n^{(1)}, \ldots, x_n^{(k-1)})^T$. Il est à remarquer que pour ce mémoire, un symbole mathématique mis en caractère gras représente un vecteur de bits, tandis que le même symbole (en caractère normal) avec comme exposant un autre symbole mis entre parenthèses représente un bit particulier du vecteur de bits. Également, remarquons que la numérotation des bits commence par 0 et va de gauche vers la droite. La fonction de transition $f : S \to S$, est représentée par la multiplication par la matrice X, dans $\mathbb{F}_2$. La transition se fait donc par

$$\mathbf{x}_n = X\mathbf{x}_{n-1} \tag{2.2}$$

où X est une matrice inversible de dimension $k \times k$ dont les éléments sont dans $\mathbb{F}_2$.

Afin d'obtenir la sortie, on a besoin d'un vecteur de L bits $\mathbf{y}_n$ que l'on définit par

$$\mathbf{z}_n = B\mathbf{x}_n \tag{2.3}$$

$$\mathbf{y}_n = Y\mathbf{z}_n \tag{2.4}$$

où $\mathbf{z}_n = (z_n^{(0)}, z_n^{(1)}, \ldots, z_n^{(k-1)})^T$, $\mathbf{y}_n = (y_n^{(0)}, y_n^{(1)}, \ldots, y_n^{(L-1)})^T$, Y est une matrice $L \times k$, B est une matrice $k \times k$ et L est appelé la *résolution de sortie du générateur*. Il faut remarquer que la matrice H introduite dans le chapitre 1 est, en fait, la matrice YB. Aux sections 4.5 et 5.5, des résultats donnent des justifications à cette décomposition. La matrice B, pour l'instant, est la matrice identité de dimension $k \times k$ (I_k). D'autres matrices B seront utilisées au chapitre 4. La matrice Y est la matrice qui décide du nombre de bits de résolution que le générateur aura à sa sortie. Si $L \leq k$, alors la matrice Y, lorsque multipliée par un vecteur, tronquera celui-ci pour ne garder que les L premiers bits. Par contre si $L > k$, alors la matrice Y aura la forme

$$Y = (I_k \;\; C)^T$$

où I_k est la matrice identité $k \times k$ et C est une matrice $(L-k) \times k$ quelconque qui permet d'avoir plus que k bits à la sortie. Soit L_m, le nombre maximum de bits disponibles par mot sur la machine utilisée. On prend $L \leq L_m$. La sortie $u_n \in [0, 1]$ associé à $\mathbf{x}_n$ est

$$u_n = \sum_{i=1}^{L} y_n^{(i-1)} 2^{-i}. \tag{2.5}$$

Les différents générateurs qui seront traités dans ce mémoire ont tous la même structure, seules leurs matrices X, Y et B respectives les différencient. Il est à remarquer que, dans la définition donnée à la section 1.2, la fonction de sortie $g : S \to U$ est la combinaison des équations (2.3), (2.4) et (2.5).

Le polynôme caractéristique des générateurs à récurrences linéaires modulo 2 est le polynôme caractéristique de la matrice X. En connaissant ce polynôme caractéristique, il est possible de vérifier si la période des états visités (les $\mathbf{x}_n$) par le générateur est maximale. En effet, si le polynôme caractéristique est primitif dans $\mathbb{F}_2$, alors on atteint la période maximale de $2^k - 1$ où k est le degré du polynôme caractéristique.

2.4 GCL Polynomial

2.4.1 Récurrences linéaires sur des espaces de polynômes

Soit $P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$, un polynôme primitif sur $\mathbb{F}_2$, et $\mathbb{F}_2[z]/(P(z))$, l'espace des polynômes dans $\mathbb{F}_2[z]$ modulo $P(z)$. Associons au vecteur de bits $\tilde{\mathbf{x}}_n = (x_n, .., x_{n+k-1})$ la série formelle de Laurent

$$\tilde{p}_n(z) = \sum_{j=1}^{\infty} x_{n+j-1} z^{-j}$$

où les $x_{n+k}, x_{n+k+1}, ..$ sont déterminés par la récurrence (2.1). On associe ensuite à $\tilde{p}_n(z)$ le polynôme (voir [12])

$$p_n(z) = P(z)\tilde{p}_n(z) \bmod 2.$$

La proposition 2.4.1 montre qu'il existe une bijection entre le vecteur $\tilde{\mathbf{x}}_n$ et le polynôme $p_n(z)$.

Proposition 2.4.1 *(L'Écuyer[12])*
L'application $\tilde{\mathbf{x}}_n \to p_n(z)$ est une bijection et satisfait

$$p_n(z) = \sum_{j=0}^{k-1} c_{n,j} z^{k-j-1}$$

où

$$\begin{pmatrix} c_{n,0} \\ c_{n,1} \\ \dots \\ c_{n,k-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ -a_1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -a_{k-1} & -a_{k-2} & \dots & -a_1 & 1 \end{pmatrix} \begin{pmatrix} x_n \\ x_{n+1} \\ \dots \\ x_{n+k-1} \end{pmatrix} \mod 2 \tag{2.6}$$

On démontre maintenant comment obtenir une récurrence permettant de passer de $p_{n-1}(z)$ à $p_n(z)$ sans avoir recours à la récurrence 2.1.

En multipliant la série $\tilde{p}_{n-1}(z)$ par z, on obtient

$$z \sum_{j=1}^{\infty} x_{(n-1)+j-1} z^{-j} = \sum_{j=0}^{\infty} x_{n+j-1} z^{-j}.$$

En enlevant le terme en z^0, on obtient $\tilde{p}_n(z)$. On écrit

$$\tilde{p}_n(z) = z\, \tilde{p}_{n-1}(z) \mod 1$$

.

En multipliant cette récurrence par $P(z)$, on obtient la récurrence

$$p_n(z) = z\, p_{n-1}(z) \mod P(z) \tag{2.7}$$

où “mod $P(z)$” signifie que $p_n(z)$ est le reste de la division par le polynôme $P(z)$ avec les opérations sur les coefficients dans $\mathbb{F}_2$. Il est maintenant clair que la récurrence (2.7) est une manière équivalente d'exprimer la récurrence (2.1).

De façon plus générale, si on désire “sauter” de s états dans la récurrence (2.7), alors on utilise

$$p_n(z) = z^s p_{n-1}(z) \mod P(z). \tag{2.8}$$

Cette équation peut être vu comme définissant un générateur à congruence linéaire (GCL) avec multiplicateur z^s et modulo $P(z)$. Pour avoir un générateur de pleine période, il est important d'avoir $\text{pgcd}(s, 2^k - 1) = 1$. Sinon, certains polynômes ne pourront jamais être atteints avec la récurrence (2.8).

2.4.2 Description du GCL polynomial avec s=1

Pour le type de générateur qui nous intéresse, nous allons utiliser la récurrence (2.8) avec $s = 1$. Ceci permet d'avoir un générateur très rapide, car l'algorithme qui permet d'implanter ce générateur ne requiert que quelques opérations rapides sur un ordinateur. Dans le cas où $s > 1$, l'implantation de ce générateur sur un ordinateur ne se fait pas aussi aisément. L'algorithme d'implantation de ce générateur est présenté au chapitre 5. On appelle le générateur de nombres aléatoires qui en résulte, avec $s = 1$, *GCL polynomial*. La récurrence du *GCL polynomial* est

$$p_n(z) = zp_{n-1}(z) \bmod P(z). \tag{2.9}$$

Afin d'être cohérent avec le cadre général défini dans la section 2.3, il est nécessaire de définir la récurrence (2.9) différemment, mais de façon équivalente. Pour ce faire, on associe au polynôme $p_n(z)$ un vecteur de k bits $\mathbf{x}_n$ où $x_n^{(i)}$ est le coefficient de z^{k-i-1} dans le polynôme $p_n(z)$. Il est important de souligner que la bijection entre $p_n(z)$ et $\mathbf{x}_n$ est complètement différente de celle établie entre $p_n(z)$ et $\tilde{\mathbf{x}}_n$ à la proposition 2.4.1. En fait, $c_{n,i} = x_n^{(i)}$ pour $i = 0, \ldots, k-1$. On définit également, de la même manière, un vecteur $\mathbf{a} = (a_1, \ldots, a_{k-1}, a_k)$ qui représente le polynôme $P(z)$. La récurrence devient alors

$$\mathbf{x}_n = X\mathbf{x}_{n-1} \tag{2.10}$$

où

$$X = \begin{pmatrix} a_1 & 1 & 0 & 0 & \ldots & 0 \\ a_2 & 0 & 1 & 0 & \ldots & 0 \\ a_3 & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k-1} & 0 & 0 & 0 & \ldots & 1 \\ a_k & 0 & 0 & 0 & \ldots & 0 \end{pmatrix}. \tag{2.11}$$

La manière d'obtenir cette équation matricielle est bien expliquée dans [20]. Afin de générer la sortie, on utilise $B = I_k$ et si $L < k$, alors $Y = I_{L\times k}$ où $I_{L\times k}$ est la matrice qui ne contient que les L premières lignes de la matrice identité I_k, sinon $Y = (I_k \;\; C)^T$ où C est une matrice $(L-k)\times k$ quelconque. La sortie u_n est obtenue par (2.5).

La période de ce type de générateur est égale à l'ordre du polynôme $P(z)$, d'où l'importance d'avoir $P(z)$ primitif. Dans ce cas, la période est $\rho = 2^k - 1$, où k est le degré du polynôme caractéristique du générateur.

2.5 Générateurs de Tausworthe

Un générateur de Tausworthe utilise explicitement la récurrence (2.1). L'état du générateur à la n-ième itération est

$$\mathbf{x}_n = (x_{ns}, x_{ns+1}, \ldots, x_{ns+k-1})$$

où s est un paramètre entier qui permet de produire plusieurs générateurs avec la même récurrence.

Le vecteur de bits de la sortie associé à cet état du générateur est

$$\mathbf{y}_n = (x_{ns}, x_{ns+1}, \ldots, x_{ns+L-1}) \tag{2.12}$$

et on utilise la sortie définie par (2.5).

On peut facilement voir que ce générateur entre bien dans le cadre général défini antérieurement. La récurrence peut être écrite sous la forme

$$\mathbf{x}_n = X^s \mathbf{x}_{n-1}$$

où

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ a_k & a_{k-1} & a_{k-2} & a_{k-3} & \dots & a_1 \end{pmatrix}.$$

Afin d'obtenir la sortie, si $L > k$, on définit la matrice $Y = (I_k \;\; C)^T$ où C permet d'obtenir les $L - w$ bits les moins significatifs de $\mathbf{y}_n$, sinon $Y = I_{L\times k}$. La matrice B, pour l'instant, est la matrice I_k.

Pour ce générateur, il n'est pas suffisant d'avoir un polynôme caractéristique primitif pour obtenir une période maximale. De façon similaire au GCL polynomial avec une valeur de s quelconque, il est aussi nécessaire d'avoir pgcd$(s, 2^k - 1) = 1$.

2.6 Générateurs TGFSR

Les GFSR peuvent être représentés par la récurrence linéaire suivante :

$$\mathbf{v}_n = \mathbf{v}_{n+m-r} \oplus \mathbf{v}_{n-r},$$

où les $\mathbf{v}_i$ sont des mots de w bits.

Dans [25], on modifie cette récurrence afin d'obtenir le *twisted GFSR* (TGFSR). Ce générateur est basé sur la récurrence

$$\mathbf{v}_n = \mathbf{v}_{n+m-r} \oplus A\mathbf{v}_{n-r}, \tag{2.13}$$

où A est une matrice binaire de dimension $w \times w$ et les $\mathbf{v}_i$ sont vus comme des vecteurs colonnes. En choisissant de façon adéquate les valeurs de w, r, m et A, il est possible d'atteindre une période de $2^{rw} - 1$, qui est la période maximale pour ce type de générateur.

Les auteurs de [25] décrivent les avantages des TGFSR sur les GFSR. On y retrouve aussi les théorèmes suivants sur les conditions nécessaires et suffisantes afin d'atteindre la période maximale.

Théorème 2.6.1 *(Matsumoto et Kurita[25])*
Soit $\phi_A(t)$, le polynôme caractéristique de la matrice A. Le générateur a une période maximale de $2^{rw}-1$ si et seulement si $\phi_A(t^r+t^m)$ est un polynôme primitif de degré rw. Dans ce cas, chaque bit du vecteur $\mathbf{v}_i$ suit une récurrence de la forme (2.1), avec polynôme caractéristique $\phi_A(t^r+t^m)$.

Il est possible de déterminer si $\phi_A(t^r+t^m)$ est primitif plus facilement grâce au prochain théorème car le polynôme pour lequel il faut vérifier la primitivité est de degré moindre.

Théorème 2.6.2 *(Matsumoto et Kurita[25])*
Soit ξ une racine de $\phi_A(t)$, alors $\phi_A(t^r+t^m)$ est primitif si et seulement si les conditions suivantes sont respectées :

(1) Le polynôme $\phi_A(t)$ est irréductible.

(2) Le polynôme $t^r+t^m+\xi$ est primitif sur $\mathbb{F}_2^w$.

On observe que ce générateur entre dans le cadre général décrit à la section 2.3, puisque

$$\begin{pmatrix} \mathbf{v}_n \\ \mathbf{v}_{n-1} \\ \vdots \\ \mathbf{v}_{n-r+2} \\ \mathbf{v}_{n-r+1} \end{pmatrix} = X \begin{pmatrix} \mathbf{v}_{n-1} \\ \mathbf{v}_{n-2} \\ \vdots \\ \mathbf{v}_{n-r+1} \\ \mathbf{v}_{n-r} \end{pmatrix} \tag{2.14}$$

où

$$X = \begin{pmatrix} & & & I_w & & A \\ I_w & & & & & \\ & I_w & & & & \\ & & I_w & & & \\ & & & \ddots & & \\ & & & & I_w & \end{pmatrix} \tag{2.15}$$

et, l'élément $(0,0)$ de la matrice I_w de la première ligne de X est l'élément $(0,(r-m-1)w)$ de X.

Il est à remarquer que l'état courant du générateur, $\mathbf{x}_n = (\mathbf{v}_n^T, \mathbf{v}_{n-1}^T, \ldots, \mathbf{v}_{n-r+1}^T)^T$, se définit sur rw bits et que c'est pour cette raison que le polynôme caractéristique est de degré rw. On peut donc déduire que la période maximale des TGFSR est de $2^{rw}-1$. Afin de générer la sortie, on a $B = I_k$ et $Y = I_{w\times k}$, car $L = w$ pour ce type de générateur. La sortie u_n est calculée selon (2.5).

2.7 Mersenne Twister

Ce générateur a été introduit pour la première fois dans [27] et est une généralisation du TGFSR de la section 2.6. La récurrence est de la forme

$$\mathbf{v}_n = \mathbf{v}_{n+m-r} \oplus A(\mathbf{v}_{n-r}^h | \mathbf{v}_{n-r+1}^b). \tag{2.16}$$

Il y a 5 paramètres à cette récurrence : les entiers r et w, un entier p, $0 \leq p \leq w-1$, un entier m, $0 \leq m \leq r$, et une matrice A dont les entrées sont dans $\mathbb{F}_2$. Les vecteurs $\mathbf{v}_n$ sont des vecteurs-colonnes. Le symbole $\mathbf{v}_{n-r}^h$ représente les $w-p$ premiers bits de $\mathbf{v}_{n-r}$ tandis que $\mathbf{v}_{n-r+1}^b$ représente les p derniers bits de $\mathbf{v}_{n-r+1}$. L'opération $|$ représente la concaténation des opérandes. Également, on observe que si $p = 0$, la récurrence (2.16) devient (2.13), d'où la généralisation du TGFSR.

On démontre maintenant que ce générateur entre dans le cadre général défini à la section 2.3. L'état du Mersenne twister est représenté sur $rw-p$ bits. La période maximale de ce type de générateur est alors de 2^{nw-p}. L'état est

$$\mathbf{x}_n = (\mathbf{v}_n^T, \mathbf{v}_{n-1}^T, \ldots, \mathbf{v}_{n-r+2}^T, \mathrm{trunc}_{w-p}(\mathbf{v}_{n-r+1})^T)^T$$

où $\mathrm{trunc}_{w-p}((\mathbf{v}_{n-r+1})^T)$ est le vecteur de $w-p$ bits composé des $w-p$ premiers bits de

$(\mathbf{v}_{n-r+1})^T$. La matrice X de dimension $(rw-p)\times(rw-p)$ est

$$X = \begin{pmatrix} & & & I_w & & & \mathrm{S} \\ I_w & & & & & & \\ & I_w & & & & & \\ & & \ddots & & & & \\ & & & & \ddots & & \\ & & & & & I_{w-p} & \end{pmatrix} \tag{2.17}$$

où

$$S = A \begin{pmatrix} & I_{w-p} \\ I_p & \end{pmatrix}$$

est une matrice $w \times w$.

Le polynôme caractéristique de la récurrence dépend de la matrice A. Un exemple d'une implantation de Mersenne twister avec son polynôme caractéristique est donné à la section 5.4.

La raison pour laquelle ce type de générateur a été développé est qu'avec les TGFSR, il est difficile d'obtenir de très longues périodes ($> 2^{2000}$, par exemple). Dans [14], on discute de l'importance d'avoir une longue période. Les algorithmes permettant de déterminer la primitivité d'un polynôme nécessitent la factorisation de $2^k - 1$. Même avec les méthodes les plus modernes, factoriser de tels nombres pour $k > 2000$ est très difficile. Par contre, si k est un exposant de Mersenne, alors la factorisation n'est pas nécessaire, puisque $2^k - 1$ est premier. Dans ce cas, il existe des algorithmes qui permettent de déterminer la primitivité du polynôme caractéristique en $\mathrm{O}(lk^2)$ opérations, où l est le nombre de termes non nuls du polynôme. De plus, à cause de la forme spéciale de la récurrence du Mersenne twister, la primitivité peut être déterminée en $\mathrm{O}(k^2)$ opérations. Dans le cas du TGFSR, $2^{rw} - 1$ ne peut jamais être premier (sauf si $w = 1$ ou $n = 1$). Par contre, $rw - p$ peut représenter n'importe quel nombre entier, y compris les exposants de Mersenne.

Afin de mieux percevoir la difficulté de trouver des polynômes primitifs pour de grandes valeurs de k, les auteurs de [27] rapportent qu'il a fallu deux semaines de calculs pour trouver un polynôme primitif de degré 19937 (19937 est un exposant de Mersenne), utilisable pour un Mersenne twister. Les auteurs ne spécifient pas le type d'ordinateur utilisé pour ces calculs.

2.8 Combinaisons de générateurs

Il est possible de combiner plusieurs générateurs afin d'obtenir de meilleures propriétés pour le générateur combiné que celles des composantes individuelles. Il s'agit de faire fonctionner les générateurs en parallèle et, à chaque itération, produire une sortie en combinant toutes les sorties de chacune des composantes.

La manière de combiner plusieurs générateurs est expliquée dans l'introduction de ce mémoire. Mais au lieu d'utiliser (1.5) pour la composante j ($j = 1, \ldots, J$), on utilise

$$\mathbf{z}_{n,j} = B_j \mathbf{x}_{n,j}$$

et

$$\mathbf{y}_{n,j} = Y_j \mathbf{z}_{n,j}.$$

On observe que $H_j = Y_j B_j$ où B_j est une matrice $k_j \times k_j$ de plein rang et Y_j est une matrice $L_j \times k_j$. Pour combiner plusieurs composantes, on utilise (1.6). En utilisant cette méthode, Tezuka [31] affirme que si les périodes de chacune des composantes sont premières entre elles, alors le générateur combiné a une période égale au produit des périodes des composantes.

La valeur de L est définie par (1.7) et la sortie u_n est déterminée par la formule habituelle (2.5). Au lieu de (1.7), on pourrait aussi utiliser

$$L = \max\{L_j \mid 1 \leq j \leq J\}. \tag{2.18}$$

Par contre, les bits les moins significatifs n'auraient pas la même période que les bits les plus significatifs. En prenant (1.7), on s'assure que tous les bits sont de période maximale. Dans le cas où on décide de prendre (2.18), il est important de compléter les vecteurs $\mathbf{y}_{n,j}$ avec $L - L_j$ bits constants puisque pour l'équation (1.6), il est nécessaire que tous les bits des $\mathbf{y}_{n,j}$ soient bien définis. Pour le besoin de ce mémoire, on utilise (1.7).

Afin de s'assurer que le générateur combiné progresse dans le cycle le plus long, où la période maximale est donnée par (1.4), il est nécessaire d'avoir $\mathbf{x}_{0,j} \neq \mathbf{0}$, pour $1 \leq j \leq J$, où $\mathbf{0}$ est le vecteur de bits ne contenant que des bits nuls. Dans le cas contraire, si pour une composante j' on a $\mathbf{x}_{0,j'} = \mathbf{0}$, alors on obtient le même générateur combiné avec les mêmes composantes sauf la composante j'.

On observe aussi que pour un générateur qui utilise k bits pour représenter son état, on pourrait s'attendre à ce que celui-ci ait une période maximale, pour le plus long cycle, de $2^k - 1$. Ce n'est pas le cas du générateur combiné, qui utilise $k = \sum_{j=1}^{J} k_j$ bits, puisque son cycle le plus long a une période de $(2^{k_1} - 1) \times \ldots \times (2^{k_J} - 1)$, qui est strictement inférieur à $2^k - 1$.

Chapitre 3

Équidistribution et façon de la calculer

Dans ce chapitre, on discute du critère le plus étudié lorsqu'on construit des générateurs utilisant une récurrence linéaire modulo 2. Ce critère, qu'on appelle *équidistribution*, vérifie si des ensembles de points produits par le générateur de nombres aléatoires occupent de façon uniforme l'espace inscrit dans des hypercubes unitaires, et ce, dans plusieurs dimensions. Il y a deux façons de procéder afin d'analyser cette uniformité : observer physiquement les ensembles de points ou analyser la récurrence pour ensuite tirer des conclusions sur l'uniformité de ces ensembles de points. C'est une approche du deuxième type que nous utilisons pour sa rapidité puisque celle-ci ne nécessite pas le calcul de tous les points. Cet avantage devient de plus en plus important à mesure que la période des générateurs à vérifier augmente. Tout ce chapitre se veut une révision, pour le lecteur, de la notation et des techniques utilisées dans [13] afin de vérifier l'équidistribution, sauf pour la dernière section qui présente une généralisation de l'équidistribution donnée dans [4].

3.1 Définition de l'équidistribution

Considérons l'ensemble Ω_t de tous les points à t dimensions produits par les valeurs successives d'un générateur, à partir de tous les états initiaux possibles $\mathbf{x}_0$, défini par

$$\Omega_t = \{\mathbf{u}_{0,t} = (u_0, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}.$$

Maintenant, supposons que l'on partitionne l'hypercube $[0,1)^t$ en $2^{t\ell}$ petits hypercubes de même grandeur. La plus grande valeur de ℓ pour laquelle chacun des petits hypercubes contient le même nombre de points est appelée la *résolution* en dimension t et est notée ℓ_t. On dit que Ω_t (et par le fait même, le générateur) est *(t,ℓ)-équidistribué* si tous les petits hypercubes contiennent le même nombre de points, soit $2^{k-t\ell}$ points. Ceci est possible seulement si $k \geq t\ell$, parce que la cardinalité de Ω_t est au plus 2^k. L'ensemble Ω_t est dit *équidistribué au maximum(ME)*, si ℓ_t atteint sa borne supérieure pour toutes les valeurs de t possibles, c'est-à-dire $\ell_t = \min(L, \lfloor k/t \rfloor)$ pour $t = 1, \ldots, k$. Un ensemble de nombres complémentaires sont les $t_\ell, 1 \leq \ell \leq L$, où t_ℓ représente la plus grande valeur de t pour lequel Ω_t est (t,ℓ)-équidistribué. On a les bornes supérieures

$$\ell_t \leq \ell_t^* \stackrel{\text{def}}{=} \min(L, \lfloor k/t \rfloor) \quad \text{pour } t = 1, \ldots, k \tag{3.1}$$

et

$$t_\ell \leq t_\ell^* \stackrel{\text{def}}{=} \lfloor k/\ell \rfloor \quad \text{pour } l = 1, \ldots, L. \tag{3.2}$$

On définit aussi *l'écart en dimension* t par $\Lambda_t \stackrel{\text{def}}{=} \ell_t^* - \ell_t$ et *l'écart en résolution* ℓ par $\Delta_l \stackrel{\text{def}}{=} t_\ell^* - t_\ell$. On dit que le générateur a une *résolution maximale en dimension* t si $\Lambda_t = 0$. De plus, celui-ci sera dit de *dimension maximale en résolution* ℓ, si $\Delta_\ell = 0$. Selon ces définitions, on peut dire que le générateur est équidistribué au maximum (ME), si $\Lambda_t = 0$ pour $t = 1, \ldots, k$ ou, de façon équivalente, si $\Delta_\ell = 0$ pour $\ell = 1, \ldots, \min(k, L)$. L'équivalence peut être vue en constatant que, si pour une valeur de t on a $\ell_t < \ell_t^*$ il s'ensuit que $\Delta_{\ell_t^*} > 0$ et que si, pour une certaine valeur de ℓ, $t_\ell < t_\ell^*$, alors $\Lambda_{t_\ell^*} > 0$.

3.2 Vérification de l'équidistribution

Dans ce qui suit, on montre la façon de vérifier l'équidistribution pour les générateurs utilisant des récurrences linéaires modulo 2. Dans cette section, on montre que le calcul du rang d'une matrice permet de trouver les valeurs de ℓ_t et de t_ℓ.

À partir des vecteurs de sortie $\mathbf{y}_n$ du générateur, construisons le vecteur ligne de $t\ell$ bits

$$\mathbf{s}_{t,\ell} = (\mathrm{trunc}_\ell(\mathbf{y}_0)^T, \ldots, \mathrm{trunc}_\ell(\mathbf{y}_{t-1})^T)$$

De (2.2),(2.3) et (2.4), on peut observer que chaque valeur de $y_n^{(i)}$ peut être exprimée comme une combinaison linéaire de $x_0^{(0)}, \ldots, x_0^{(k-1)}$. On obtient

$$y_n^{(i)} = \sum_{g=0}^{k-1} b_{n,i,g} x_0^{(g)} \tag{3.3}$$

où $b_{n,i,g} \in \mathbb{F}_2$. Alors $\mathbf{s}_{t,\ell}$ peut être exprimé par

$$\mathbf{s}_{t,\ell} = \mathbf{x}_0^T B_{t,\ell}, \tag{3.4}$$

où $B_{t,\ell}$ est une matrice $k \times t\ell$, avec éléments dans $\mathbb{F}_2$, qui peut être définie comme suit : si $y_n^{(i)}$ est le g-ième élément de $\mathbf{s}_{t,\ell}$, alors la g-ième colonne de la matrice est $(b_{n,i,0}, \ldots, b_{n,i,k-1})^T$. La proposition suivante donne une condition nécessaire et suffisante pour que le générateur soit (t,ℓ)-équidistribué.

Proposition 3.2.1 *(L'Écuyer[13])*
Le générateur est (t,ℓ)-équidistribué si et seulement si la matrice $B_{t,\ell}$ est de plein rang.

Afin de trouver les valeurs des $b_{n,i,g}$, nous utilisons directement l'implantation du générateur. Supposons que nous ayons comme état initial $\mathbf{x}_0 = (x_0^{(0)}, \ldots, x_0^{(k-1)})^T = \mathbf{e}_q^T$, le q-ième vecteur unitaire en dimension k, alors l'équation (3.3) se réduit à $y_n^{(i)} = b_{n,i,q}$. On peut ensuite compléter le vecteur $\mathbf{s}_{t,\ell}$ avec l'implantation même du générateur. En utilisant le générateur, on peut construire facilement la matrice $B_{t,\ell}$ avec l'algorithme 3.2.1.

Algorithme 3.2.1 *(L'Écuyer[13])*

1. $q \leftarrow 0$
2. $\mathbf{x}_0 \leftarrow \mathbf{e}_q$
3. *Utiliser l'implantation du générateur pour trouver le vecteur* $\mathbf{s}_{t,\ell}$
4. *Mettre le vecteur* $\mathbf{s}_{t,\ell}$ *courant à la ligne* q *de la matrice* $B_{t,\ell}$
5. $q \leftarrow q + 1$
6. *Retourner à l'étape 2 si* $q < k$*, sinon arrêter.*

Il est à remarquer que les générateurs qui utilisent le cadre général, avec $B = I_k$, sont toujours $(1, \min(k, L))$-équidistribués par définition de Ω_t, même avec une matrice X quelconque. Un générateur $(1, \min(k, L))$-équidistribué signifie que toutes les $2^{\min(k,L)}$ valeurs possibles de $\text{trunc}_{\min(k,L)}(\mathbf{y}_n)$ sont observées si l'on considère tous les cycles possibles.

3.3 Vérifier l'équidistribution pour des générateurs combinés

On peut combiner plusieurs générateurs afin d'améliorer l'équidistribution comparativement à l'équidistribution des composantes individuelles. On peut ainsi combiner J générateurs avec l'équation (1.6). De l'équation (3.3), on obtient pour chaque composante j, $1 \leq j \leq J$,

$$y_{n,j}^{(i)} = \sum_{g=0}^{k_j - 1} b_{n,j,i,g} x_{0,j}^{(g)}$$

où $b_{n,j,i,g} \in \mathbb{F}_2$. Le vecteur de bits

$$\mathbf{s}_{j,t,\ell} = (\text{trunc}_\ell(\mathbf{y}_{0,j})^T, \ldots, \text{trunc}_\ell(\mathbf{y}_{t-1,j})^T)$$

peut être exprimé par

$$\mathbf{s}_{j,t,\ell} = \mathbf{x}_{0,j}^T B_{j,t,\ell}$$

où $B_{j,t,\ell}$ est une matrice à éléments binaires de dimension $k_j \times t\ell$. Ses éléments peuvent être calculés avec les méthodes décrites dans la section précédente en considérant chaque composante indépendamment.

On peut observer que

$$y_n^{(i)} = y_{n,1}^{(i)} + \ldots + y_{n,J}^{(i)} = \sum_{j=1}^{J} \sum_{g=0}^{k_j-1} b_{n,j,i,g} y_{0,j}^{(i)}$$

pour chaque valeur de n et i possible. On obtient alors

$$\mathbf{s}_{t,\ell} = \sum_{j=1}^{J} \mathbf{s}_{j,t,\ell} = \sum_{j=1}^{J} \mathbf{x}_{0,j}^T B_{j,t,\ell}$$

que l'on peut réécrire de la façon suivante :

$$\mathbf{s}_{t,\ell} \equiv \tilde{\mathbf{x}}_0^T \tilde{B}_{t,\ell}$$

où $\tilde{B}_{t,\ell}$ est une matrice $k \times t\ell$ définie par la juxtaposition verticale des $B_{j,t,\ell}$ et où $\tilde{\mathbf{x}}_0^T$ est obtenu en juxtaposant les $\mathbf{x}_{0,j}^T$ horizontalement. On obtient une proposition similaire à celle de la section précédente, sauf que celle-ci est plus générale. Comme dans le cas du générateur simple, il est possible de déterminer l'équidistribution du générateur en trouvant le rang d'un matrice. Si chacun des $\mathbf{x}_{j,n}$ n'ont pas le même nombre de bits de précision, alors la valeur de L, pour la vérification de l'équidistribution, est donnée par (1.7).

Proposition 3.3.1 *(L'Écuyer[13]) Un générateur combiné est (t, ℓ)-équidistribué si et seulement si la matrice $\tilde{B}_{t,\ell}$ est de plein rang $t\ell$.*

3.4 Conditions suffisantes pour l'équidistribution maximale

Afin de vérifier si un générateur est ME, on peut calculer les valeurs ℓ_t pour $t = 1, \ldots, k$ ou t_ℓ pour $\ell = 1, \ldots, \min(k, L)$. Mais il n'est pas nécessaire de calculer toutes les valeurs de ℓ_t ou de t_ℓ puisqu'une (t, ℓ)-équidistribution implique une (t', ℓ')-équidistribution pour tous $t' \le t$ et $\ell' \le \ell$. Ceci peut être vu à l'aide de la matrice $\tilde{B}_{t,\ell}$. Si le générateur est (t, l)-équidistribué, alors la matrice $\tilde{B}_{t,\ell}$ est de plein rang $t\ell$. Pour savoir si le générateur est $(t-1, \ell)$-équidistribué, on calcule le rang de la matrice $\tilde{B}_{t-1,\ell}$. Mais, il est à remarquer que l'on peut obtenir la matrice $\tilde{B}_{t-1,\ell}$ en enlevant les ℓ dernières colonnes de $\tilde{B}_{t,\ell}$. Alors puisque $\tilde{B}_{t,\ell}$ est de rang $t\ell$, $\tilde{B}_{t-1,\ell}$ est de rang $(t-1)\ell$. Un raisonnement semblable peut être fait pour déduire le rang de $\tilde{B}_{t,\ell-1}$. Également, si $\ell_{t'} = \ell_t$ pour $t' < t$ et si une résolution maximale est obtenue en dimension t, alors elle est aussi obtenue en dimension t'. De façon similaire, si $t_{\ell'} = t_\ell$ pour $\ell' < \ell$ et le générateur est de dimension maximale pour la résolution ℓ, alors il est aussi de dimension maximale pour la résolution ℓ'. La prochaine proposition nous dicte pour quelles valeurs de t et de ℓ on doit vérifier l'équidistribution. Mais d'abord, définissons les ensembles suivants :

$$
\begin{aligned}
\Phi_1 &= \{\max(2, \lfloor k/L \rfloor), \ldots, \lfloor \sqrt{k} \rfloor\}, \\
\Phi_2 &= \{t = \lfloor k/\ell \rfloor \mid \ell = 1, \ldots, \lfloor \sqrt{k-1} \rfloor\}, \\
\Psi_1 &= \{1, \ldots, \lfloor \sqrt{k} \rfloor\} \\
\text{et} & \\
\Psi_2 &= \{\ell = \lfloor k/t \rfloor \mid t = \max(2, \lfloor k/L \rfloor), \ldots, \lfloor \sqrt{k-1} \rfloor\},
\end{aligned}
$$

Proposition 3.4.1 *(L'Écuyer[13]) Un générateur avec période maximale est ME si et seulement si $\Lambda_t = 0$ pour tout $t \in \Phi_1 \cup \Phi_2$, si et seulement si $\Delta_\ell = 0$ pour tout $\ell \in \Psi_1 \cup \Psi_2$.*

3.5 Vérification de l'équidistribution des projections

Dans la section 3.1, on a discuté de l'ensemble de points Ω_t et de la signification de l'équidistribution du générateur générant cet ensemble de points. Cet ensemble de points est généré par les valeurs successives produites par le générateur. L'équation (3.5) définit un critère d'uniformité plus rigoureux que la simple équidistribution et tente de mesurer l'uniformité d'ensembles de points lorsqu'on considère des projections dans des dimensions non successives. L'utilité de ce critère est bien illustré dans [4, 22] pour l'intégration quasi-Monte Carlo : afin de réduire l'erreur d'intégration, il est préférable de vérifier l'équidistribution dans plusieurs ensembles de points formés par des t-tuplets de valeurs non successives du générateur. Dans [4], on définit le critère 3.5 qui est une généralisation du critère d'équidistribution et que l'on nomme *équidistribution des projections*. C'est de ce critère dont il est question dans cette section.

On définit

$$\tilde{\Delta}_{t_1,\ldots,t_d} = \max\left(\max_{I=\{1,\ldots,s\},s=1,\ldots,t_1} (\ell^*_{|I|}(n) - \ell_I), \max_{I\in S(t_s,s),s=2,\ldots,d} (\ell^*_{|I|}(n) - \ell_I) \right), \tag{3.5}$$

où $S(t_s, s) = \{\{i_1, \ldots, i_s\}, 1 = i_1 < \ldots < i_s \leq t_s\}$. La quantité $\ell^*_{|I|}(n)$ est donnée par $\lfloor k/|I| \rfloor$ et correspond à la résolution maximale pour un ensemble de $n = 2^k$ points dans $[0, 1)^{|I|}$. L'ensemble I est un ensemble d'indices de dimensions.

La première partie du critère, soit

$$\max_{I=\{1,\ldots,s\},s=1,\ldots,t_1} (\ell^*_{|I|}(n) - \ell_I),$$

détermine l'écart maximal entre la résolution maximale, $\ell^*_{|I|}(n)$, et la résolution obtenue, ℓ_I, en prenant les ensembles de points produits par des s-tuplets de valeurs successives du générateur, soit

$$\{\mathbf{u}_s = (u_0, \ldots, u_{s-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}.$$

pour $s = 1, \ldots, t_1$.

Une autre façon d'exprimer la première partie est par

$$\max_{t=1,\ldots,t_1} \Lambda_t.$$

La deuxième partie du critère, soit

$$\max_{I\in S(t_s,s),s=2,\ldots,d} (\ell^*_{|I|}(n) - \ell_I),$$

détermine également l'écart maximal entre la résolution maximale et la résolution obtenue mais en prenant des ensembles de points de valeurs non successives. Ces ensembles de points sont

$$\{\mathbf{u} = (u_{i_1}, \ldots, u_{i_s}) : \mathbf{x}_0 \in \mathbb{F}_2^k\},$$

pour $s = 2, \ldots, d$ et $1 = i_1 < \ldots < i_s \leq t_s$. Le paramètre t_s permet de choisir, pour les s-tuplets, la valeur maximale de i_s et par le fait même, du point de vue pratique, de contrôler le nombre d'ensembles de points à vérifier. La valeur $\tilde{\Delta}_{t_1,\ldots,t_d}$ est la valeur maximale de $\ell^*_{|I|}(n) - \ell_I$ qu'on obtient dans les deux parties.

Il est important d'observer que ce critère $\tilde{\Delta}_{t_1,\ldots,t_d}$ généralise la propriété ME parce que le générateur est ME si et seulement si $\tilde{\Delta}_k = 0$. Dans ce cas, on obtient

$$\tilde{\Delta}_k = \max_{I=\{1,\ldots,s\},s=1,\ldots,k} (\ell^*_{|I|}(n) - \ell_I) = \max_{t=1,\ldots,t_1} \Lambda_t.$$

Dans la définition que l'on a donnée d'un générateur ME, on dit qu'il est ME si $\Lambda_t = 0$ pour $t = 1, \ldots, k$. Puisque $\Lambda_t \geq 0$, alors l'équivalence entre $\tilde{\Delta}_k = 0$ et la propriété ME devient évidente.

Chapitre 4

Transformations linéaires modifiant la sortie d'un générateur

Selon notre définition d'un générateur de nombres aléatoires de la section 1.2, $G = (S, s_0, f, U, g)$, il nous est toujours possible de modifier la fonction de sortie g. La fonction $g : S \to U$ permet d'associer à chaque état du générateur un élément de U.

Ce cadre général nous permet une liberté qui, jusqu'à présent dans ce mémoire, n'a pas été exploitée : la matrice B. On va maintenant utiliser cette matrice B dans un but très simple : **obtenir la meilleure équidistribution possible**.

Pour ce mémoire, on s'intéresse à des matrices B de dimension $k \times k$ dont les $k' = \min(k, L)$ premières lignes sont linéairement indépendantes. Ceci permet d'obtenir un générateur (1,k')-équidistribué. Lors de la vérification de l'équidistribution, on n'aura pas besoin de vérifier l'équidistribution en 1 dimension puisque celle-ci sera garantie. Dans le cas où $B = I_k$, la (1,k')-équidistribution est garantie par définition de Ω_t. Par contre, quand $B \neq I_k$, la (1,k')-équidistribution n'est pas garantie. La proposition suivante donne une condition suffisante pour que le générateur soit (1,k')-équidistribué.

Proposition 4.0.1 *Si les* $\min(k, L)$ *premières lignes de* B *sont linéairement indépendantes,* *alors le générateur sera (1,*$\min(k,L)$*)-équidistribué.*

preuve :

En construisant la matrice $B_{1,\min(k,L)}$ *avec l'algorithme 3.2.1, on obtient* $B_{1,\min(k,L)} = \tilde{B}$ *où* $\tilde{B}$ *est une matrice* $k \times \min(k,L)$ *qui contient les* $\min(k,L)$ *premières colonnes de* B^T*. Puisque les* $\min(k,L)$ *premières lignes de* B *sont linéairement indépendantes, alors il est évident que* $B_{1,\min(k,L)} = \tilde{B}$ *est au moins de rang* $\min(k,L)$*. Selon la proposition 3.2.1, le générateur est alors (1,*$\min(k,L)$*)-équidistribué.*■

4.1 Transformations linéaires sur des générateurs combinés

Dans cette sous-section, on explique la manière d'appliquer des transformations linéaires sur des générateurs combinés, tout en gardant la $(1,k')$-équidistribution. Soit $k'_j = \min(k_j, L_j)$. Il y a deux façons principales d'appliquer les transformations linéaires :

1. Sur chacune des composantes de façon indépendante, tel qu'expliqué à la section 2.8, c'est-à-dire avoir une transformation B_j propre à chacune des composantes.
2. À la sortie du générateur combiné. La matrice Y_j est une matrice $L_j \times k_j$ qui fait en sorte que $\mathbf{y}_{n,j}$ soit un vecteur de L_j bits (si $L_j > k_j$, alors Y_j est donc de la forme $(I_{k_j}\ C_j)^T$ où C_j sert à rajouter les $L_j - k_j$ bits manquants, sinon $Y_j = I_{L_j \times k_j}$). Si on définit

   $$L_{\min} = \min_{j=1,\dots,J} L_j,$$

   alors la matrice B est une matrice $\min(k, L_{\min}) \times \min(k, L_{\min})$ et la matrice Y est une matrice $L \times k$ où $L \leq L_{\min}$. On procède ainsi :

   $$\begin{aligned}
   \mathbf{z}_{n,j} &= \mathbf{x}_{n,j}, \quad 1 \leq j \leq J \\
   \mathbf{y}_{n,j} &= Y_j \mathbf{z}_{n,j}, \quad 1 \leq j \leq J \\
   \mathbf{z}_n &= B\ \mathrm{trunc}_{L_{\min}}(\oplus_{j=1}^{J} \mathbf{y}_{n,j}) \\
   \mathbf{y}_n &= Y \mathbf{z}_n
   \end{aligned}$$

Pour la première option, par la proposition 4.0.1, afin que le générateur soit $(1, \min(k,L))$-équidistribué, il est nécessaire que les $\min(k,L)$ premières lignes des matrices B_j soient linéairement indépendantes car en fait on obtient $\mathbf{z}_n$ par

$$\mathbf{z}_n = (\tilde{B}_1 \ \ldots \ \tilde{B}_J)(\mathbf{x}_{n,1}^T, \ \ldots \ , \mathbf{x}_{n,J}^T)^T$$

où les $\tilde{B}_j$ sont des matrices $L \times k_j$, qui ne contiennent que les L premières lignes de B_j.

De façon similaire, pour la deuxième option, le vecteur de bits $\mathbf{z}_n$ est obtenu en effectuant $B\tilde{B}(\mathbf{x}_{n,1}^T, \ldots, \mathbf{x}_{n,J}^T)^T$ où

$$\tilde{B} = \begin{pmatrix} \tilde{Y}_1 & \tilde{Y}_2 & \ldots & \tilde{Y}_J \end{pmatrix}$$

et les matrices $\tilde{Y}_j$ sont des matrices $L_{\min} \times k_j$ qui ne contiennent que les $L_{\min}$ premières lignes de Y_j. Afin que le générateur soit $(1, \min(k, L))$-équidistribué, il faut que les $\min(k, L)$ premières lignes de la matrice $B\tilde{B}$ soient linéairement indépendantes.

La première option a l'avantage d'avoir des composantes qui sont $(1, k_j')$-équidistribuées si elles sont prises individuellement. La seconde a l'avantage d'offrir des générateurs plus rapides (car on n'effectue qu'une seule transformation linéaire, au lieu de J).

4.2 La permutation des coordonnées

La première transformation linéaire B abordée dans ce mémoire est la permutation des coordonnées. Celle-ci, que l'on note $B = P$, consiste à changer la position des bits du vecteur $\mathbf{x}_n$ pour obtenir le vecteur $\mathbf{z}_n$ de la façon suivante :

$$z_n^{(i)} = x_n^{(\pi(i))}, \quad 0 \leq i < L - 1 \tag{4.1}$$

$$\tag{4.2}$$

où $\pi : \mathbb{Z}_k \to \mathbb{Z}_k$ est une permutation. Une forme acceptable de permutation qui est utilisée plus tard dans ce mémoire est celle où $\pi(i) = pi + q \bmod k$ pour $0 \leq i < k$. Cette forme est notée $P_{p,q}$.

Une autre forme de cette transformation linéaire est celle où $\pi(i) = pi + q \bmod w$ pour $0 \leq i < w$ et $\pi(i) = i$ pour $w < i < k$. Elle correspond à permuter les w premiers bits du vecteur d'état et à laisser les $k - w$ suivants aux mêmes positions. Cette forme est surtout utilisée pour les générateurs TGFSR, car avec ces générateurs, l'état est découpé en r vecteurs de w bits. Appliquer une permutation, lors de l'implantation, sur les w premiers bits de l'état est plus facile que de faire la permutation à l'intérieur du vecteur d'état au complet. Cette forme est notée $\tilde{P}^{w}_{p,q}$.

4.3 Le self-tempering

Cette transformation linéaire a été développée afin d'améliorer les propriétés statistiques des GCL polynomiaux. En effet, lorsque que $\mathbf{x}_{n-1}$ a peu de bits non nuls, étant donnée la récurrence très simple du générateur, alors il est très probable que $\mathbf{x}_n$ aura aussi très peu de bits non nuls. Ce fait important amène des problèmes de corrélation entre les valeurs successives u_n si $B = I_k$. La stratégie prise par le self-tempering est une heuristique qui tente de remédier à ce problème en faisant en sorte que $\mathbf{z}_n$ ait beaucoup de bits non nuls, si $\mathbf{x}_n$ en a peu. De cette manière, la tendance à avoir peu de bits pour plusieurs valeurs successives est réduite. Il est important de remarquer que le nombre moyen de bits non nuls par sortie ne change pas avec cette transformation. Cette transformation linéaire, quoique conçue pour les GCL polynomiaux, peut être utilisée pour n'importe quel type de générateur de nombres aléatoires qui utilise une récurrence linéaire modulo 2.

Le self-tempering a deux paramètres, c et d, ayant les contraintes $0 < d < c \leq k$. On note cette transformation par $B = S_{c,d}$. Si on note $K = \lceil k/c \rceil$, alors les opérations suivantes décrivent ce que fait $S_{c,d}$ pour obtenir $\mathbf{z}_n$.

Algorithme 4.3.1 *Self-tempering* $S_{c,d}$

1. $\mathbf{b}_j = (\mathbf{x}_n \ll c(j-1))\ \&\ \mathbf{m}_c$ *pour* $j = 1, \ldots, K$
2. $\mathbf{e} = (\bigoplus_{j=1}^{K} \mathbf{b}_j) \ll d$
3. $\mathbf{d}_j = \mathbf{b}_j \oplus \mathbf{e}$ *pour* $j = 1, \ldots, K$
4. $\mathbf{z}_n = \bigoplus_{j=1}^{K} (\mathbf{d}_j \gg c(j-1))$

Il est à noter que $\ll d$ signifie un décalage de d bits vers la gauche et $\gg d$ signifie un décalage de d bits vers la droite. L'opération $\mathbf{x}\ \&\ \mathbf{y}$ signifie qu'on effectue une multiplication de $\mathbf{x}$

par $\mathbf{y}$ modulo 2, bit par bit (voir table 2.2).

Pour cette transformation, $\mathbf{m}_c$ est un masque qui ne conserve que les c premiers bits. Cette transformation fait ce qui suit : À l'étape 1, on découpe le vecteur $\mathbf{x}_n$ en K vecteurs $\mathbf{b}_j$ de c bits. À la deuxième étape de l'algorithme, on fait un ou-exclusif de tous les $\mathbf{b}_j$ et on décale le résultat de d bits vers la gauche pour mettre le résultat dans le vecteur $\mathbf{e}$ de c bits. Ensuite, on reprend chacun des vecteurs $\mathbf{b}_j$ et on effectue un ou-exclusif avec le vecteur $\mathbf{e}$, ce qui produit les vecteurs $\mathbf{d}_j$. La dernière étape consiste à juxtaposer les vecteurs $\mathbf{d}_j$ pour obtenir $\mathbf{z}_n$

Cette transformation est facilement implantée sur un ordinateur 32 bits. Dans ce cas on prend $c = 32$. Afin de représenter un mot de k bits $\mathbf{x}_n$, on aura besoin de $K = \lceil k/32 \rceil$ mots de 32 bits : $\mathbf{x}_n^1, \mathbf{x}_n^2, .., \mathbf{x}_n^K$. Donc $\mathbf{x}_n$ est la juxtaposition des $\mathbf{x}_n^j$. Dans ce cas particulier, $S_{32,d}$ se fait de la manière suivante :

Algorithme 4.3.2 *Self-tempering* $S_{32,d}$

1. $\mathbf{e} = (\bigoplus_{j=1}^{K} \mathbf{x}_n^j) \ll d$
2. $\mathbf{z}_n^j = \mathbf{x}_n^j \oplus \mathbf{e}$ *pour* $j = 1, \ldots, K-1$.
3. $\mathbf{z}_n^K = (\mathbf{x}_n^K \oplus \mathbf{e}) \ \& \ \mathbf{m}_h$

où $h = k - 32(K-1)$, $\mathbf{e}$ est un vecteur de 32 bits et le résultat de la transformation $S_{32,d}$ sur $\mathbf{x}_n$ est le vecteur de bits $\mathbf{z}_n$ qui est la juxtaposition horizontale des $\mathbf{z}_n^j$ $(1 \leq j \leq K)$.

En analysant la transformation linéaire donné par les algorithmes 4.3.1 et 4.3.2, on se rend compte que dans le meilleur des cas, s'il y a peu de bits non nuls dans $\mathbf{x}_n$, alors le nombre de bits non nuls est multiplié par $K+1$. En moyenne, le nombre de bits non nuls reste le même.

4.4 Le tempering de Matsumoto-Kurita

Le *tempering de Matsumoto-Kurita* est une transformation linéaire qui a été introduite par Matsumoto et Kurita dans [26] afin d'améliorer l'équidistribution des TGFSR. Avec ce type de générateur, on a toujours $L \leq k$. Le tempering de Matsumoto-Kurita utilisé dans [26] se définit par cette combinaison d'opérations :

Algorithme 4.4.1 *Tempering de Matsumoto-Kurita*

1. $\tilde{\mathbf{x}}_n = \; trunc_L(\mathbf{x}_n)$
2. $\mathbf{r}_n = \tilde{\mathbf{x}}_n \oplus \; ((\tilde{\mathbf{x}}_n \ll \; \eta) \; \& \; \mathbf{b})$
3. $\mathbf{y}_n = \mathbf{r}_n \oplus \; ((\mathbf{r}_n \ll \; \mu) \; \& \; \mathbf{c})$.

où $\mathbf{r}_n$ est un vecteur de L bits et les paramètres η et μ sont des entiers tels que $0 \leq \eta$, $\mu \leq L-1$; $\mathbf{b}$ et $\mathbf{c}$ sont des masques de L bits. Maintenant, si nous voulons appliquer cette transformation linéaire, comment faire pour choisir les paramètres $\eta, \mu, \mathbf{b}$ et $\mathbf{c}$? Dans [26], il est recommandé de choisir μ près de $\lfloor L/2 \rfloor - 1$ et η près de $\mu/2$. Les auteurs ont trouvé qu'avec ces valeurs de η et μ, ils peuvent atteindre assez facilement une $(\lfloor k/2 \rfloor, 2)$-équidistribution. Pour trouver $\mathbf{b}$ et $\mathbf{c}$, ils utilisent l'algorithme décrit dans la section 4.4.1.

À la première opération du tempering de Matsumoto-Kurita, on tronque le vecteur $\mathbf{x}_n$ pour ne garder que les L premiers bits, L étant la résolution de la sortie du générateur. On se restreint donc à n'utiliser que l'information contenue dans les L premiers bits de l'état $\mathbf{x}_n$. Si $L < k$, il serait avantageux d'utiliser plus que L bits. Cette idée en tête, on modifie le tempering de Matsumoto-Kurita pour obtenir la généralisation suivante :

Algorithme 4.4.2 *Tempering de Matsumoto-Kurita généralisé*

1. $\overline{\mathbf{x}}_n = \; trunc_w(\mathbf{x}_n)$
2. $\mathbf{r}_n = \overline{\mathbf{x}}_n \oplus \; ((\overline{\mathbf{x}}_n \ll \; \eta) \; \& \; \mathbf{b})$
3. $\mathbf{z}_n = \mathbf{r}_n \oplus \; ((\mathbf{r}_n \ll \; \mu) \; \& \; \mathbf{c})$

On note le résultat de ces étapes par $\mathbf{z}_n = T_{w,\eta,\mu,\mathbf{b},\mathbf{c}}\mathbf{x}_n$ et c'est celle-ci que l'on appelle tempering de Matsumoto-Kurita pour le reste du mémoire. Cette nouvelle transformation linéaire permet de contrôler le nombre de bits qui est utilisé par la transformation linéaire par le paramètre w. Il faut cependant que $w \leq k$. La règle générale est que $w \leq k$ pour les générateurs de type GCL polynomial et de Tausworthe et pour les TGFSR et les Mersenne twister, w est de même valeur que le paramètre w des générateurs. Ces choix sont dictés pour des raisons de simplicité lors de l'implantation des générateurs.

S'inspirant des recommandations données dans [26], il a été observé que des valeurs de μ près de $\lfloor w/2 \rfloor - 1$ et des valeurs de η près de $\mu/2$ donnent de bons résultats. L'algorithme de la prochaine sous-section permet de trouver, pour des valeurs de η et de μ données, des valeurs de $\mathbf{b}$ et $\mathbf{c}$ qui donnent une bonne équidistribution au générateur.

La différence majeure entre les algorithmes 4.4.1 et 4.4.2 est le nombre de bits utilisés par la transformation linéaire. Avec l'algorithme 4.4.1, on effectue le tempering de Matsumoto-Kurita sur les L premiers bits de l'état du générateur, tandis qu'avec l'algorithme 4.4.2, on peut choisir le nombre de bits de l'état qui seront utilisés pour la transformation linéaire par l'intermédiaire du paramètre w. Par exemple, supposons que pour un générateur donné $k = 200$ et $L = 32$. En prenant $w = k = 200$ dans l'algorithme 4.4.2, on utilise tous les bits de l'état afin de produire la sortie. En prenant $w = L = 32$, comme dans l'algorithme 4.4.1, on n'utilise que les 32 premiers bits de l'état et les 168 derniers bits n'influencent nullement la sortie.

4.4.1 Algorithme d'optimisation du tempering Matsumoto-Kurita

Supposons qu'en vérifiant l'équidistribution d'un générateur auquel on applique un tempering de Matsumoto-Kurita, on observe qu'il est (t^*_ℓ, ℓ)-équidistribué pour $\ell = 1, \ldots, \ell' - 1$, mais pas pour $\ell = \ell'$. En analysant la transformation linéaire, on observe que les bits des masques $\mathbf{b}$ et $\mathbf{c}$ qui affectent le bit $y_n^{(\ell'-1)}$ qui empêche le générateur d'être $(t^*_{\ell'}, \ell')$-équidistribué sont les bits $b^{(\ell'-1)}, b^{(\ell'+\mu-1)}$ et $c^{(\ell'-1)}$. La figure 4.1 permet de visualiser ceci. Dans le diagramme, on regarde les rectangles horizontaux comme des vecteurs de bits, où le bit 0 est à la gauche du rectangle et le bit $w - 1$ est à la droite. Également, on voit les opérations en fonction du vecteur de bits $\overline{\mathbf{x}}_n$, puisque $\mathbf{r}_n = \overline{\mathbf{x}}_n \oplus ((\overline{\mathbf{x}}_\mathbf{n} \ll \eta) \,\&\, \mathbf{b})$. Il y a un rectangle vertical en pointillés qui indique bien les bits qui affectent le bit $y_n^{(\ell'-1)}$. Les symboles "$\langle 1 \rangle$", "$\langle 2 \rangle$", "$\langle 3 \rangle$" et "$\langle 4 \rangle$" représentent les bits $y_n^{(\ell'-1)}$, $c^{(\ell'-1)}$, $b^{(\ell'+\mu-1)}$ et $b^{(\ell'-1)}$, respectivement.

Montrons maintenant comment améliorer la $(t^*_{\ell'}, \ell')$-équidistribution du générateur en changeant les valeurs de $b^{(\ell'-1)}, b^{(\ell'+\mu-1)}$ et $c^{(\ell'-1)}$. Le diagramme montre bien que si $\ell' \leq$

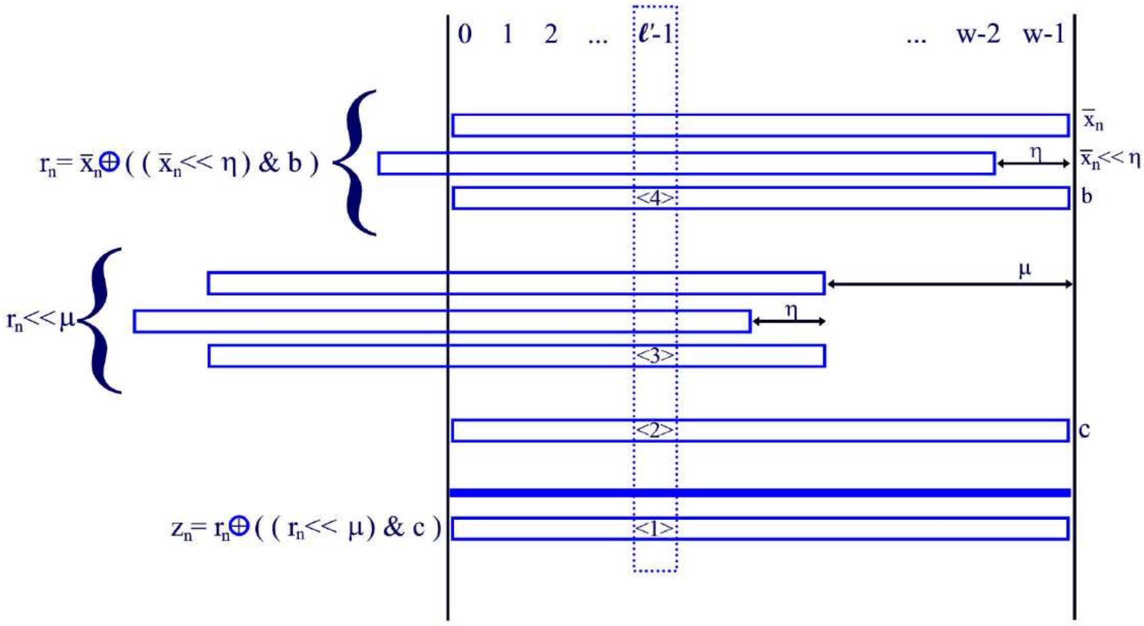

Figure 4.1 – Illustration du tempering

$w - \mu$, alors le bit $c^{(\ell'-1)}$ affecte le bit ${y_n}^{(\ell'-1)}$. Dans le cas où $\ell' \le w - \mu$, on a le choix de choisir 0 ou 1 comme valeur de $c^{(\ell'-1)}$, si $\ell' > w - \mu$, on met ce bit à zéro. On voit également que si $\ell' - \mu > 0$ et $c^{(\ell'-\mu-1)} = 1$, alors le bit $b^{(\ell'-1)}$ affecte le bit ${y_n}^{(\ell'-\mu-1)}$. Dans ce cas, on ne doit pas changer $b^{(\ell'-1)}$ pour ne pas nuire à la $(t^*_{(\ell'-1)}, \ell'-1)$-équidistribution. Sinon, si $\ell' \le w - \eta$, alors le bit $b^{(\ell'-1)}$ affecte aussi le bit ${y_n}^{(\ell'-1)}$. Dans ce cas, on peut mettre le bit $b^{(\ell'-1)}$ à 0 ou à 1. Dans l'autre cas, on met le bit $b^{(\ell'-1)}$ à zéro. On observe aussi que si $\ell' + \mu \le w - \eta$ et si $c^{(\ell'-1)} = 1$, alors le bit $b^{(\ell'+\mu-1)}$ affecte le bit ${y_n}^{(\ell'-1)}$. Dans le cas où $\ell' + \mu \le w - \eta$ et si $c^{(\ell'-1)} = 1$, alors on peut mettre le bit $b^{(\ell'+\mu-1)}$ à 0 ou à 1, sinon on met le bit $b^{(\ell'+\mu-1)}$ à zéro.

Ces observations faites grâce à la figure 4.1 permettent de construire l'algorithme 4.4.3 qui est suggéré dans [26]. Cet algorithme est conçu pour des générateur à une seule composante. Un algorithme, s'inspirant de celui-ci, permettant du tempering sur plusieurs composantes est décrit dans la section 4.4.2. Avant d'amorcer l'algorithme 4.4.3, il est important de déterminer les valeurs de δ_v pour $v = 1, \ldots, k'$. Ces valeurs représentent les valeurs de Δ_v que l'on peut accepter et permettent de contrôler la qualité de l'équidistribution des générateurs que l'on désire. Ainsi, si on veut des générateurs qui sont ME, alors on mettra $\delta_v = 0$ pour $v = 1, \ldots, k'$.

L'algorithme 4.4.3 construit un arbre de recherche. À chaque noeud, on associe cinq valeurs : v, Δ_v, $b^{(v-1)}$, $b^{(v+\mu-1)}$ et $c^{(v-1)}$. Si pour un noeud donné où $v = \tilde{v}$, on observe que $\Delta_v \leq \delta_v$, alors celui aura des enfants avec $v \leftarrow \tilde{v}+1$ et chacun de ceux-ci aura une configuration différente des bits $b^{(v-1)}$, $b^{(v+\mu-1)}$ et $c^{(v-1)}$. Si $\Delta_v > \delta_v$, alors ce noeud n'aura pas d'enfants. Voici l'algorithme de façon plus formelle.

Algorithme 4.4.3 *(Matsumoto et Kurita[26])*

1. *Fixer η et μ et mettre $\mathbf{b} \leftarrow \mathbf{0}$, et $\mathbf{c} \leftarrow \mathbf{0}$.*
2. *On trouve ℓ', la plus grande valeur de ℓ telle que le générateur, avec $\mathbf{b}$ et $\mathbf{c}$, est (t^*_ℓ, ℓ)-équidistribué, pour $\ell = 1, \ldots, \ell'-1$. $v \leftarrow \ell'$.*
3. *Le racine de l'arbre de recherche est le noeud où $v = \ell'$.*
4. *Pour chaque noeud tel que $v = \tilde{v}-1$, celui-ci aura au plus huit enfants où chaque enfant correspond à une configuration des bits $b^{(\tilde{v}-1)}$, $b^{(\tilde{v}+\mu-1)}$ et $c^{(\tilde{v}-1)}$. Pour chacun des enfants, vérifier si l'équidistribution est telle que $\Delta_{\tilde{v}} > \delta_{\tilde{v}}$. Si c'est le cas, alors celui-ci n'aura pas d'enfants qui lui sont propres. Sinon, il aura des enfants avec une valeur de $v \leftarrow \tilde{v}+1$. Voici les règles qui permettent de choisir les configurations de bits pour chaque enfant du noeud avec $v = \tilde{v}-1$:*

$$c^{(\tilde{v}-1)} \quad \begin{cases} = 0 & si\ \tilde{v} > w - \mu \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.3}$$

$$b^{(\tilde{v}-1)} \quad \begin{cases} sans\ changement & si\ c^{(\tilde{v}-\mu-1)} = 1\ et\ \tilde{v}-\mu > 0 \\ = 0 & si\ \tilde{v} > w - \eta \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.4}$$

$$b^{(\tilde{v}+\mu-1)} \quad \begin{cases} = 0 & si\ \tilde{v}+\mu > w - s\ ou\ c^{(\tilde{v}-1)} = 0 \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.5}$$

5. *Explorer l'arbre en profondeur et arrêter l'algorithme lorsqu'on a trouvé un générateur qui soit tel que $\Delta_v \leq \delta_v$ pour $v = 1, \ldots, k'$ ou lorsque l'on a exploré tout l'arbre.*

Voici quelques explications sur cet algorithme. Dans les règles de sélection des bits $b^{(\tilde{v}-1)}$, $b^{(\tilde{v}+\mu-1)}$ et $c^{(\tilde{v}-1)}$, il y a plusieurs façons des choisir ceux-ci qui sont valides. Ainsi, lorsqu'il est écrit par exemple, "$b^{(\tilde{v}-1)} \in \mathbb{F}_2$ sinon", ceci indique que les deux valeurs de $\mathbb{F}_2$ sont possibles pour ce bit si les autres conditions ne sont pas respectées. En analysant les règles, on se rend vite compte qu'il peut y avoir jusqu'à huit configurations des bits $b^{(\tilde{v}-1)}$, $b^{(\tilde{v}+\mu-1)}$ et $c^{(\tilde{v}-1)}$. Aussi, l'algorithme spécifie de rechercher l'arbre en profondeur puisque le résultat voulu n'est jamais dans les premiers niveaux de l'arbre de recherche mais plutôt dans le dernier niveau possible.

4.4.2 Algorithme d'optimisation du tempering de Matsumoto-Kurita pour les générateurs combinés

L'algorithme suivant est l'algorithme d'optimisation du tempering de Matsumoto-Kurita qui a été développé afin de tenir compte des générateurs combinés. Ce nouvel algorithme a l'avantage de modifier le tempering de Matsumoto-Kurita sur chacune des composantes afin d'optimiser l'équidistribution du générateur combiné. Parmi les deux façons décrites dans la section 4.1 d'appliquer les transformations linéaires, il s'agit d'une combinaison du premier type car cet algorithme suppose que l'on applique un tempering de Matsumoto-Kurita différent sur chacune des composantes. Si on désire faire un seul tempering de Matsumoto-Kurita à la sortie du générateur combiné, alors on peut utiliser l'algorithme 4.4.3.

Pour la composante j, les paramètres du tempering de Matsumoto-Kurita sont w_j, μ_j, η_j, $\mathbf{b}_j$ et $\mathbf{c}_j$. L'ensemble des composantes pour lesquelles on désire optimiser le tempering de Matsumoto-Kurita est Θ (Il se peut que l'on désire optimiser le tempering pour certaines composantes et, pour d'autres, on désire utiliser un tempering de Matsumoto-Kurita avec des paramètres fixés à l'avance. Il se peut également que, sur certaines composantes, il n'y ait pas de tempering de Matsumoto-Kurita du tout).

Algorithme 4.4.4 *Optimisation du tempering de Matsumoto-Kurita pour les générateurs combinés*

1. *Mettre* $\mathbf{b}_j \leftarrow \mathbf{b}_j^0$, $\mathbf{c}_j \leftarrow \mathbf{c}_j^0$, *pour* $j \in \Theta$. $k' = \min\{k'_j \mid 1 \leq j \leq J\}$.

2. *Vérifier l'équidistribution du générateur combiné. On constate que celui-ci, avec les* $\mathbf{b}_j$ *et* $\mathbf{c}_j$ *courants, est tel que* $\Delta_\ell \leq \delta_\ell$, *pour* $\ell = 1, \ldots, \ell' - 1$.

3. *Si* $\ell' - 1 = k'$, *alors on arrête car on a une équidistribution telle que* $\Delta_v \leq \delta_v$ *pour* $v = 1, \ldots, k'$.

4. $v \leftarrow \ell'$.

5. $n_v \leftarrow 0$.

6. *Si* $v = \ell' - 1$, *on arrête l'algorithme (on n'a pas obtenu l'équidistribution voulue), sinon on peut changer les masques* $\mathbf{b}_j$ *et* $\mathbf{c}_j$ $(j \in \Theta)$ *selon les restrictions suivantes :*

$$c_j^{(v-1)} \quad \begin{cases} 0 & si\ v > w_j - \mu_j \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.6}$$

$$b_j^{(v-1)} \quad \begin{cases} sans\ changement & si\ c_j^{(v-\mu_j-1)} = 1\ et\ v - \mu_j > 0 \\ 0 & si\ v > w_j - \eta_j \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.7}$$

$$b_j^{(v+\mu_j-1)} \quad \begin{cases} 0 & si\ v + \mu_j > w_j - \eta_j\ ou\ c_j^{(v-1)} = 0 \\ \in \mathbb{F}_2 & sinon \end{cases} \tag{4.8}$$

On peut choisir ces valeurs de façon systématique ou au hasard. Cet aspect est discuté plus loin.

7. $n_v \leftarrow n_v + 1$.

8. *On vérifie la* (t_v, v)*-équidistribution.*

   Si on a $\Delta_v \leq \delta_v$ *et* $v < k'$, *on retourne à l'étape 5 avec* $v \leftarrow v + 1$,

   Si on a $\Delta_v \leq \delta_v$ *et* $v = k'$ *alors aller à l'étape 10.*

   Si on n'a pas $\Delta_v \leq \delta_v$ *et* $n_v > N(v)$, *alors aller à l'étape 9.*

   Sinon on recommence l'étape 6 avec la même valeur de v *mais en obtenant une autre configuration des bits* $b_j^{(v-1)}, b_j^{(v+\mu_j-1)}$ *et* $c_j^{(v-1)}$.

9. *Recommencer l'étape 6 avec* $v \leftarrow v - 1$.

10. *On a atteint une équidistribution telle que* $\Delta_v \leq \delta_v$ *pour* $v = 1, \ldots, k'$ *pour ce générateur avec les valeur de* $\mathbf{b}_j$ *et* $\mathbf{c}_j$ *obtenues.*

Dans cet algorithme, $N(v)$ est le nombre de configurations possibles des bits $b_j^{(v-1)}$, $b_j^{(v+\mu_j-1)}$ et $c_j^{(v-1)}$, $j \in \Theta$ et le nombre n_v contient le nombre de configurations que l'on a essayées avec cette valeur de v. Si on ne tient pas compte des restrictions données à l'étape 6, il y a $2^{3|\Theta|}$ configurations possibles de ces $3|\Theta|$ bits. Mais, en fonction des restrictions, il se peut qu'il y en ait moins. À l'étape 1, on initialise $\mathbf{b}_j$ et $\mathbf{c}_j$ avec des vecteurs de bits aléatoires $\mathbf{b}_j^0$ et $\mathbf{c}_j^0$ pour les composantes $j \in \Theta$. Les valeurs η_j et μ_j sont choisies au début de l'algorithme et ne changent pas.

L'algorithme peut se voir comme une recherche en profondeur dans un arbre, comme dans l'algorithme 4.4.3, où chaque nœud au niveau v a $N(v)$ branches et le paramètre n_v indique le nombre de branches explorées par niveau. Le noeud supérieur est le noeud $v = \ell'$. À chaque noeud, il est nécessaire de vérifier l'équidistribution, ce qui implique la diagonalisation d'une matrice de dimension $k \times \lfloor k/v \rfloor v$. Ceci est la raison majeure de la difficulté de trouver des bons masques $\mathbf{b}_j$ et $\mathbf{c}_j$ étant donné l'effort de calcul demandé pour diagonaliser ces matrices. Par contre, on remarque que $B_{\lfloor k/(v-1) \rfloor, v-1}$ et $B_{\lfloor k/v \rfloor, v}$ ont des colonnes communes. Si on enregistre les opérations que l'on a effectuées sur $B_{\lfloor k/(v-1) \rfloor, v-1}$ et on les applique sur $B_{\lfloor k/v \rfloor, v}$, alors le travail de diagonalisation sera moins grand. Cette stratégie n'a pu être étudiée dans le cadre de ce mémoire, mais elle pourrait permettre de réduire le temps de calcul pour la diagonalisation des matrices. En fait, on pourrait utiliser cette stratégie pour la vérification de l'équidistribution en général et non seulement dans le contexte de l'algorithme 4.4.4.

À l'étape 6, on doit modifier la valeur des bits $b_j^{(v-1)}, b_j^{(v+\mu_j-1)}$ et $c_j^{(v-1)}$. Si on n'obtient pas $\Delta_v \leq \delta_v$ pour une configuration de bits, on doit en essayer d'autres jusqu'à ce qu'on obtienne $\Delta_v \leq \delta_v$. On peut essayer les configurations de façon systématique, c'est-à-dire essayer toutes les $N(v)$ configurations possibles. On peut également essayer les configurations de façon aléatoire. Soit $N'(v) < N(v)$, le nombre de configurations à essayer pour chaque valeur de v. En essayant les configurations de façon aléatoire et en choisissant des valeurs de $N'(v)$ plus petites que $N(v)$, on parcourt un arbre de recherche plus restreint. Ceci permet de ne pas s'attarder trop longtemps sur de mauvaises branches. En effet, on a pu remarquer qu'en remontant dans l'arbre et en redescendant par d'autres branches, il est possible d'obtenir une meilleure équidistribution que si nous avions persisté dans la première branche. Il a été observé qu'en mettant $N'(v)$ décroissant puis croissant avec v, on obtient de bons résultats. Ce choix de $N'(v)$ se justifie par l'heuristique suivante :

- On veut beaucoup de noeuds près de la racine pour pouvoir essayer plusieurs vecteurs $\mathbf{b}_j$ et $\mathbf{c}_j$;
- si une branche s'avère mauvaise et v est assez grand ($N'(v)$ est petit), alors on passe moins de temps à l'explorer, car les chances de progression sont basses ;
- si une branche est bonne et v est assez grand, même si $N'(v)$ est petit, les chances de progression sont grandes ;
- quand v est près de L ($N'(v)$ est grand), on est près du but, donc on explore le plus possible afin d'arriver au noeud $v = L$.

Exemple Recherche aléatoire

Soit un générateur à une composante. On désire explorer l'arbre de façon aléatoire. On initialise les valeurs de $N'(v)$:

$$N'(1) = 6,\ N'(2) = 5,\ N'(3) = 2,\ N'(4) = 3,\ N'(5) = 4,\ N'(6) = 5,\ N'(7) = 7,\ N'(8) = 8$$

On observe que $N'(v)$ croît puis décroît. On obtient un arbre de recherche plus restreint que pour la recherche systématique, mais qui a de bonnes chances de trouver une solution, s'il en existe une. Par exemple, il est impossible d'avoir un TGFSR à une composante qui soit ME (ce qui montré dans le prochain chapitre). Si on initialise $\delta_\ell = 0$ pour $\ell = 1, \ldots, L$, alors il n'existe pas de solution.

Une modification que l'on peut apporter à l'algorithme 4.4.4 est le remplacement de l'étape 2 par $\ell' \leftarrow 1$. Cette modification permet de parcourir un arbre plus étendu. Dans l'algorithme 4.4.4, on suppose que la racine en haut de l'arbre de recherche est une bonne racine et que l'on doit construire notre arbre basé sur cette racine. Avec cette modification, on ne suppose rien et on commence la recherche avec la valeur de v la plus basse possible, afin de ne pas rendre certaines configurations des bits $\mathbf{b}$ et $\mathbf{c}$ inaccessibles pour l'arbre de recherche.

4.5 Composition de transformations linéaires

On peut effectuer plusieurs transformations linéaires sur la sortie d'une composante d'un générateur. Donc B sera de la forme $B = B_1 B_2 \ldots B_m$ où m est le nombre de transformations linéaires que l'on veut appliquer. Si chacune des transformations est de plein rang, alors la composition sera aussi de plein rang. Souvent, en pratique on observe qu'il est possible d'obtenir une bonne équidistribution en effectuant plusieurs transformations linéaires consécutives sur le même vecteur d'état afin d'obtenir le vecteur de sortie. La seule restriction qui rend cette alternative peu attirante est le ralentissement provoqué pour la génération des nombres aléatoires sur un ordinateur.

Si l'on désire optimiser le tempering de Matsumoto-Kurita sur une composante avec l'algorithme 4.4.4, il est préférable que celui-ci soit la dernière transformation à être effectuée sur la composante. La raison est simple, supposons que nous ayons une $(t^*_{(v-1)}, v-1)$-équidistribution à un certain point dans l'algorithme 4.4.4, alors les bits des masques $\mathbf{b}_j$ et $\mathbf{c}_j$ empêchant une (t^*_v, v)-équidistribution ne seraient plus les bits $b_j^{(v-1)}, b_j^{(v+\mu_j-1)}$ et $c_j^{(v-1)}$, mais plutôt d'autres bits plus difficiles à retracer.

On peut remarquer également qu'il est inutile d'appliquer plusieurs permutations consécutives puisque l'on peut obtenir le même résultat avec une seule permutation. En effet, par exemple si $\pi(i) = pi + q \bmod k$ pour $i < k$ alors on obtient

$$\begin{aligned} \hat{z}_n^{(i)} &= x_n^{(p_1 i + q_1 \bmod k)} \\ {z_n}^{(i)} &= \hat{z}_n^{(p_2 i + q_2 \bmod k)} \\ &= x_n^{(p_1(p_2 i + q_2) + q_1 \bmod k)} \\ &= x_n^{(p_1 p_2 i + p_1 q_2 + q_1 \bmod k)}. \end{aligned}$$

Ce qui démontre bien que P_{p_1,q_1} suivit de P_{p_2,q_2} est l'équivalent de $P_{p,q}$ où $p \equiv p_1 p_2 \bmod k$ et $q \equiv p_1 q_2 + q_1 \bmod k$.

Chapitre 5

Implantation des générateurs

Ce chapitre présente quelques nouveaux algorithmes qui ont été développés dans le cadre de ce mémoire de maîtrise. Parmi ceux-ci, on retrouve la généralisation de QuickTaus et l'implantation de permutations particulières pour les GCL polynomiaux et les TGFSR. Les références sont donnés pour les algorithmes qui existent déjà dans la littérature.

5.1 Tausworthe

Dans [13], on présente l'algorithme Quicktaus, qui permet d'implanter un générateur de Tausworthe, comme le laisse entendre son nom, rapidement. Cet algorithme fonctionne si $P(z)$ est un trinôme, $P(z) = z^k - z^q - 1$, et les paramètres du générateur vérifient les conditions $0 < 2q \leq L$ et $0 < s < k - q$ et $\text{pgcd}(s, 2^k - 1) = 1$. Cet algorithme permet de calculer, L bits à la fois, la récurrence. Dans ce qui suit, nous présentons un nouvel algorithme qui généralise Quicktaus afin de ne pas se limiter à un trinôme.

Soit $P(z) = z^k - z^{q_1} - \cdots - z^{q_m} - 1$, un polynôme primitif dans $\mathbb{F}_2$ tel que $q_1 > q_2 > \ldots > q_m$. On définit $r = k - q_1$. Les vecteurs $\mathbf{a}$, $\mathbf{b}$, $\mathbf{b}_1$, ..., $\mathbf{b}_m$ et $\mathbf{c}$ sont des vecteurs de L bits. Au début de l'algorithme, le vecteur $\mathbf{a}$ contient $\mathbf{y}_{n-1}$ et, à la fin, contient $\mathbf{y}_n$ (à noter que l'on suppose que $B = I_k$). Le vecteur $\mathbf{c}$ contient un masque composé de k uns suivis de

$L - k$ zéros.

Algorithme 5.1.1 *Algorithme Quicktaus généralisé*

1. *Pour* i *de* 1 *à* m

   $\mathbf{b}_i \leftarrow \mathbf{a} \ll q_i$

2. $\mathbf{b} \leftarrow \bigoplus_{i=1}^{m} \mathbf{b}_i$

3. $\mathbf{b} \leftarrow \mathbf{a} \oplus \mathbf{b}$

4. $\mathbf{b} \leftarrow \mathbf{b} \gg (k - s)$

5. $\mathbf{a} \leftarrow \mathbf{a} \,\&\, \mathbf{c}$

6. $\mathbf{a} \leftarrow \mathbf{a} \ll s$

7. $\mathbf{a} \leftarrow \mathbf{a} \oplus \mathbf{b}$

Pour les détails de chacune des étapes de cet algorithme, voir [13]. Les étapes 1 et 2 de l'algorithme généralisé correspondent à l'étape 1 de l'algorithme Quicktaus. Les autres étapes sont les mêmes.

Comme dans le cas de Quicktaus, le vecteur $\mathbf{a}$ doit être initialisé correctement, c'est-à-dire, avec une valeur de $\mathbf{y}_0$ où ses bits suivent la récurrence. Les k premiers bits de $\mathbf{y}_0$ peuvent être choisis de façon arbitraire, sauf tous des zéros. Les autres bits de $\mathbf{y}_0$ doivent être déterminés par la récurrence (2.1). L'algorithme suivant permet de compléter $\mathbf{y}_0$, r bits à la fois. Les k premiers bits de $\mathbf{a}$ sont initialisés arbitrairement et les $L - k$ suivants sont initialement nuls.

Algorithme 5.1.2 *Procédure d'initialisation pour Quicktaus généralisé*

1. *Pour* i *de* 1 *à* m

   $\mathbf{b}_i \leftarrow \mathbf{a} \ll q_i$

2. $\mathbf{b} \leftarrow \bigoplus_{i=1}^{m} \mathbf{b}_i$

3. $\mathbf{b} \leftarrow \mathbf{a} \oplus \mathbf{b}$

4. $\mathbf{b} \leftarrow \mathbf{b} \gg k$

5. $\mathbf{a} \leftarrow \mathbf{a} \oplus \mathbf{b}$

On doit répéter cette procédure d'initialisation autant de fois que nécessaire afin de remplir les L bits de $\mathbf{a}$. Une fois cette procédure appliquée $\lceil \frac{L-k}{r} \rceil$ fois, $\mathbf{a}$ est le vecteur $\mathbf{y}_0$ initialisé correctement.

On explique maintenant une façon d'appliquer une transformation linéaire sur un générateur de Tausworthe quand $L > k$ avec cette implantation. On sait que $Y = (I_k \quad C)^T$ dans ce cas. On remarque aussi que l'implantation passe de $\mathbf{y}_{n-1}$ à $\mathbf{y}_n$ directement. Si on appliquait une transformation $B \neq I_k$, alors la matrice H aurait la forme $H = (B \quad BC)^T$. Cette matrice H peut être à nouveau décomposée pour obtenir $H = Y_0 B$ où $Y_0 = (I_k \quad C_0)$ et $C_0 = BC$. Au lieu de choisir C comme décrit à la section 2.5, on choisit C de telle sorte que $\mathbf{r} = BC\mathbf{x}_n = (x_{ns+k}, \ldots, x_{ns+L-1})$. On obtient l'algorithme suivant.

Algorithme 5.1.3 *Transformation linéaire sur un générateur de Tausworthe*

1. $\mathbf{x}_n \leftarrow \mathrm{trunc}_k(\mathbf{y}_n)$
2. $\mathbf{r} \leftarrow \mathrm{trunc}_{L-k}(\mathbf{y}_n \ll k)$
3. $\mathbf{z}_n = B\mathbf{x}_n$
4. $\mathbf{y}_n = \mathbf{x}_n | \mathbf{r}$

Puisque l'algorithme 5.1.1 ne manipule pas les vecteurs $\mathbf{x}_n$, il faut aller les extraire à chaque itération. Ceci est obtenu par l'opération $\mathbf{x}_n$=$\mathrm{trunc}_k(\mathbf{y}_n)$. On conserve les $L-k$ derniers bits de $\mathbf{y}_n$ avec l'opération $\mathbf{r}$=$\mathrm{trunc}_{L-k}(\mathbf{y}_n \ll k)$. On peut maintenant effectuer $\mathbf{z}_n = B\mathbf{x}_n$ et obtenir le nouveau $\mathbf{y}_n$ par $\mathbf{y}_n = \mathbf{x}_n | \mathbf{r}$.

5.2 GCL polynomial

La façon d'implanter ce générateur est la suivante. Pour passer d'un état à l'autre, on utilise la récurrence (2.10). En partant d'un état initial, soit le polynôme $p_0(z) = \sum_{j=0}^{k-1} x_0^{(j)} z^{k-j}$, qui est représenté par le vecteur de bits $\mathbf{x}_n$, on progresse dans la récurrence en multipliant $\mathbf{x}_n$ par la matrice X définie par (2.11). De façon concrète, la multiplication est effectuée par l'algorithme 5.2.1 dont l'idée vient de Raymond Couture.

Algorithme 5.2.1 *GCL polynomial*

Si $x_{n-1}^{(0)} = 1$

$$\mathbf{x_n} \leftarrow (\mathbf{x_{n-1}} \ll 1) \oplus \mathbf{a}$$

sinon

$$\mathbf{x_n} \leftarrow \mathbf{x_{n-1}} \ll 1$$

5.3 TGFSR

Une fois les valeurs de r, m, w et A choisies on peut implanter l'algorithme de la façon suivante.

Algorithme 5.3.1 *TGFSR (Matsumoto et Kurita[26])*

1. $l \leftarrow 0$, $n \leftarrow 0$
2. *Initialiser l'état du générateur, soit* $\mathbf{v}_0$, $\mathbf{v}_1$, .., $\mathbf{v}_{r-1}$, *de manière à ce que ses bits ne soient pas tous nuls.*
3. *Produire comme sortie* $\mathbf{y}_n \leftarrow \mathbf{v}_l$ *(si* $B = I_k$*)*
4. $\mathbf{v}_l \leftarrow \mathbf{v}_{(l+m) \mod r} \oplus A\mathbf{v}_l$
5. $l \leftarrow (l+1) \mod r$, $n \leftarrow n+1$
6. *Aller à l'étape 3.*

Il est important de choisir une matrice A dont l'opération $A\mathbf{v}_n$ est facile à calculer. La matrice A qui sera utilisée pour le reste de ce mémoire est la matrice

$$A = R = \begin{pmatrix} & & & & a_0 \\ 1 & & & & a_1 \\ & 1 & & & a_2 \\ & & \ddots & & \\ & & & 1 & a_{w-1} \end{pmatrix}. \tag{5.1}$$

Il y a deux raisons principales pour ce choix de A. La première est que $\mathbf{y} \leftarrow A\mathbf{v}_n$ peut être calculé facilement par l'algorithme suivant où $\mathbf{a} = (a_0, a_1, \ldots, a_{w-1})$.

Algorithme 5.3.2 *Multiplication de* $\mathbf{y} = R\mathbf{v}_n$ *(Matsumoto et Kurita[25])*

Si $v_n^{(w-1)} = 1$

$\mathbf{y} \leftarrow (\mathbf{v}_n \gg 1) \oplus \mathbf{a}$

sinon

$\mathbf{y} \leftarrow \mathbf{v}_n \gg 1$

La deuxième raison est qu'avec ce choix de A, on obtient directement le polynôme caractéristique :

$$\phi_A(t) = t^w + \sum_{i=0}^{w-1} a_i t^i,$$

ce qui facilite les calculs permettant de connaître le polynôme caractéristique de la récurrence et, du fait même, la période du générateur.

Lorsqu'on recherche des paramètres r, m, w et $\mathbf{r}$, la proposition suivante peut nous aider.

Proposition 5.3.1 *(Matsumoto et Kurita[25]) Soit* $\eta \in \mathbb{F}_2^w$. *Le polynôme* $t^r + t^m + \eta$ *dans* $\mathbb{F}_2^w$ *a un nombre pair de facteurs irréductibles. Il n'est pas irréductible si une des conditions suivantes est vérifiée :*

(1) $r \equiv \pm 3 \; mod \; 8$, w *impair*, m *pair et* $(2r \; mod \; m) \not\equiv 0$

(2) $r \equiv \pm 3 \; mod \; 8$, w *impair*, m *impair et* $(2r \; mod \; r - m) \not\equiv 0$

(3) r *pair*, m *pair*

(4) r *pair*, $r \neq 2m$ *et* $rm \equiv 0, 2 \; mod \; 8$

(5) r *pair*, $r \neq 2m, rm \equiv 4, 6 \; mod \; 8$ *et* w *est pair*

Pour un générateur dont l'état est représenté sur $k = rw$ bits, on sait par 3.2 que t_ℓ^* est une borne supérieure sur la dimension maximale pour laquelle le générateur est équidistribué avec une résolution de ℓ bits. Pour les TGFSR, il existe une borne supérieure beaucoup plus restreignante que t_ℓ^*. La proposition suivante donne cette borne supérieure pour les TGFSR construits avec une matrice A générale.

Proposition 5.3.2 *(Matsumoto et Kurita[26]) Pour un TGFSR avec matrice A quelconque, alors on a*

$$r \text{ divise } t_\ell \textit{ et } t_\ell \le \tilde{t}_\ell = r\lfloor w/\ell \rfloor \quad (v = 1, 2, \ldots, w)$$

Il existe un problème pour tous les TGFSR qui utilisent la matrice A de l'équation 5.1 : ils ne sont que $(r, 2)$-équidistribués. C'est-à-dire qu'ils n'atteignent même pas la borne $\tilde{t}_\ell$. C'est pour atteindre cette borne que le tempering de Matsumoto-Kurita a été développé. En appliquant une transformation linéaire B inversible à la sortie du générateur avec comme matrice $A = R$, on obtient la récurrence

$$\mathbf{y}_n = \mathbf{y}_{n+m-r} + BRB^{-1}\mathbf{y}_{n-r}.$$

Cette nouvelle récurrence est aussi un TGFSR mais dont la matrice A est BRB^{-1}. Le tempering de Matsumoto-Kurita est une transformation B qui permet aux TGFSR d'atteindre la borne $\tilde{t}_\ell$. Un liste de tels TGFSR est donnée dans [26].

5.4 Mersenne twister

Dans cette section, on décrit l'algorithme, tel que présenté dans [27], qui permet d'implanter un générateur Mersenne twister. Avant de définir l'algorithme, définissons certains symboles. La matrice A et le vecteur $\mathbf{a}$ sont les mêmes que ceux de la section 5.3. Le symbole $\mathbf{u}$ est un vecteur de bits composé de $w - p$ uns suivis de p zéros, tandis que $\mathbf{d}$ est un vecteur de bits composé de $w - p$ zéros suivis de p uns. Les $\mathbf{v}_j$ sont des vecteurs de w bits.

Algorithme 5.4.1 *Mersenne Twister (Matsumoto et Nishimura [27])*

1. $l \leftarrow 0$, $n \leftarrow 0$
2. *Initialiser l'état, soit* $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{r-1}$*, de manière à ce que ses bits ne soient pas tous nuls.*
3. $\mathbf{w} \leftarrow (\mathbf{v}_l \,\&\, \mathbf{u}) \oplus (\mathbf{v}_{(l+1) \bmod r} \,\&\, \mathbf{d})$
4. $\mathbf{v}_l \leftarrow \mathbf{v}_{(l+m) \bmod r} \oplus A\mathbf{w}$

5. *Produire* $\mathbf{y}_n \leftarrow \mathbf{v}_l$ *comme sortie si* $(B = I_k)$.

6. $l \leftarrow (l+1) \bmod r$, $n \leftarrow n+1$

7. *Aller à l'étape 3.*

Pour cet algorithme, la multiplication $A\mathbf{w}$ se fait par l'algorithme 5.3.2 et le polynôme caractéristique de la récurrence est

$$\begin{aligned} \phi_X(t) &= (t^r + t^m)^{w-p}(t^{r-1} + t^{m-1})^p + \sum_{i=0}^{p-1} a_i (t^r + t^m)^{w-p}(t^{r-1} + t^{m-1})^{p-i-1} \\ &\quad + \sum_{i=p}^{w-1} a_{w-i-1}(t^r + t^m)^{w-i-1}. \end{aligned} \tag{5.2}$$

5.5 Récurrence dans l'espace des vecteurs de sortie

Dans cette section, nous démontrons comment faire évoluer la récurrence (2.2) dans l'espace des vecteurs $\mathbf{z}_n = B\mathbf{x}_n$ sans jamais à avoir à calculer les vecteurs d'états $\mathbf{x}_n$ pour n'importe quel générateur qui entre dans le cadre général défini au chapitre 2. Pour ce faire, il est important que la matrice B soit inversible. Dans ce cas

$$\mathbf{z}_n = B\mathbf{x}_n = BX\mathbf{x}_{n-1} = BXB^{-1}\mathbf{z}_{n-1} = Q\mathbf{z}_{n-1}$$

Ceci démontre qu'il est possible de faire évoluer directement la récurrence d'un état transformé à un autre état transformé. Il n'est donc plus nécessaire de produire les valeurs de $\mathbf{x}_n$ et obtenir $\mathbf{z}_n$ en appliquant B à $\mathbf{x}_n$. La récurrence qu'on utilise est maintenant $\mathbf{z}_n = Q\mathbf{z}_{n-1}$ où $Q = BXB^{-1}$.

5.5.1 GCL polynomial avec permutation des coordonnées $B = P_{p,q}$

Dans cette sous-section, nous introduisons un nouvel algorithme qui permet d'implanter un type de permutation sur un GCL polynomial. Pour le type de permutation considéré, la matrice des permutations $P_{p,q}$ est inversible. Dans ce cas, on doit calculer certaines quantités afin de connaître la matrice Q. Soit $\tilde{\mathbf{a}} = P_{p,q}\mathbf{a}$, α est tel que $p\alpha \equiv 1 \bmod k$, β est tel que $p\beta + q \equiv k-1 \bmod k$ et γ est tel que $p\gamma + q \equiv 0 \bmod k$. La multiplication par la matrice Q se fait par l'algorithme 5.5.1.

Algorithme 5.5.1 *GCL polynomial avec $P_{p,q}$*

Si $z_{n-1}^{(\gamma)} = 1$

$$\mathbf{z}_n = ((\mathbf{z}_{n-1} \tilde{\lll} \alpha) \,\&\, \mathbf{d}_\beta) \;\oplus\; \tilde{\mathbf{a}}$$

sinon

$$\mathbf{z}_n = (\mathbf{z}_{n-1} \tilde{\lll} \alpha) \,\&\, \mathbf{d}_\beta$$

Le masque $\mathbf{d}_\beta$ met le bit β à zéro et $\mathbf{x}\tilde{\lll}\alpha$ signifie un décalage rotatif de α bits vers la gauche du vecteur de bit $\mathbf{x}$ (si $\mathbf{x}$ est un vecteur de L bits, alors on a que $\mathbf{x}\tilde{\lll}j = (\mathbf{x} \lll j) \oplus (\mathbf{x} \ggg L-j)$).

Proposition 5.5.1 *Une itération de l'algorithme 5.5.1 est équivalente à une itération de l'algorithme 5.2.1 suivie d'une permutation $P_{p,q}$.*

Preuve :

D'abord, il est important de remarquer que la fonction $\pi^{-1}(j)$ est l'inverse de $\pi(i)$ et $\pi^{-1}(j) = p^{-1}j - p^{-1}q$ où $pp^{-1} \bmod k \equiv 1$.

Pour cette preuve, on définit le résultat de $P_{p,q}\mathbf{x}_n$ par $\tilde{\mathbf{z}}_n$. Puisque $P_{p,q}$ ne fait que changer l'ordre des bits, chaque bit $x_n^{(j)}$ a la même valeur que le bit $\tilde{z}_n^{(\pi^{-1}(j))}$. Le bit $x^{(0)}$ correspond au bit $\tilde{z}^{(\pi^{-1}(0))}$ et par définition $\pi^{-1}(0) = \gamma$. On obtient que la condition "Si $x_{n-1}^{(0)} = 1$" est équivalente à "Si $\tilde{z}_{n-1}^{(\gamma)} = 1$".

L'opération $\mathbf{x}_n = \mathbf{x}_{n-1} \lll 1$ fait en sorte que le bit à la position j de $\mathbf{x}_{n-1}$ se retrouve à la position $j-1$ dans $\mathbf{x}_n$ pour $j = 1, \ldots, k-1$. Si on applique $P_{p,q}$ à $\mathbf{x}_{n-1}$ et $\mathbf{x}_n$, alors on remarque que le bit $\pi^{-1}(j)$ de $\tilde{\mathbf{z}}_{n-1}$ se retrouve à la position $\pi^{-1}(j-1)$ dans le vecteur $\mathbf{z}_n$. Il se trouve que

$$\pi^{-1}(j) - \pi^{-1}(j-1) = p^{-1}j - p^{-1}q - p^{-1}(j-1) + p^{-1}q \bmod k = p^{-1} \bmod k.$$

Par définition, $p^{-1} \bmod k = \alpha$. On obtient que le bit i de $\tilde{\mathbf{z}}_{n-1}$ est à la position $i - \alpha \bmod k$

dans $\tilde{\mathbf{z}}_n$*, pour* $i \in \{\pi^{-1}(j)|j = 1, \ldots, k-1\}$.

L'opération $\mathbf{x}_n = \mathbf{x}_{n-1} \ll 1$ *fait également en sorte que le bit* $j = k-1$ *de* $\mathbf{x}_n$ *est 0. Si l'on applique* $P_{p,q}$ *à* $\mathbf{x}_n$*, on remarque que le bit* $\tilde{z}_n^{(\pi^{-1}(k-1))}$ *est zéro. On a, par définition, que* $\beta = \pi^{-1}(k-1)$.

On peut maintenant affirmer que l'opération

$$\mathbf{x}_n = \mathbf{x}_{n-1} \ll 1$$

suivie de $P_{p,q}\mathbf{x}_n$ *correspond bien à*

$$\tilde{\mathbf{z}}_n = ((\tilde{\mathbf{z}}_{n-1} \lll \alpha) \,\&\, \mathbf{d}_\beta).$$

L'opération $\mathbf{x}_n \oplus \mathbf{a}$ *suivie de* $P_{p,q}$ *correspond à l'opération* $\tilde{\mathbf{z}}_n \oplus \tilde{\mathbf{a}}$ *puisque* $P_{p,q}(\mathbf{x}_n \oplus \mathbf{a}) = \tilde{\mathbf{z}}_n \oplus \tilde{\mathbf{a}}$.

Quand l'algorithme 5.2.1 est suivi d'une multiplication par $P_{p,q}$*, alors il existe pour chacune des opérations de l'algorithme une opération correspondante qui obtient le même résultat sans utiliser les vecteurs* $\mathbf{x}_n$ *et* $\mathbf{x}_{n-1}$*. Il suffit de substituer les opérations correspondantes et* $\tilde{\mathbf{z}}_n$ *et* $\tilde{\mathbf{z}}_{n-1}$ *par* $\mathbf{z}_n$ *et* $\mathbf{z}_{n-1}$ *afin d'obtenir l'algorithme 5.5.1.*■

5.5.2 TGFSR avec permutation des coordonnées $\tilde{P}^w_{p,q}$

De façon similaire au GCL polynomial, avec le TGFSR, il est possible d'implanter une permutation des coordonnées de la forme $\tilde{P}_{p,q}$ sans avoir à calculer le vecteur d'états. On obtient la récurrence 2.13

$$\mathbf{w}_n = \mathbf{w}_{n+m-r} + \tilde{A}\mathbf{w}_{n-r}$$

où $\tilde{A} = \tilde{P}^w_{p,q} A \tilde{P}^{w\,T}_{p,q}$ et $\mathbf{w}_n = \tilde{P}^w_{p,q}\mathbf{v}_n$. Dans ce contexte, on modifie l'algorithme du TGFSR de la manière suivante :

Algorithme 5.5.2 *TGFSR avec* $\tilde{P}^w_{p,q}$

1. $l \leftarrow 0$, $n \leftarrow 0$
2. *Initialiser* $\mathbf{w}_0, \mathbf{w}_1, .., \mathbf{w}_{r-1}$ *de manière à ce qu'ils ne soient pas tous nuls.*
3. *Produire comme sortie* $\mathbf{y}_n = \mathbf{w}_l$ *(s'il n'y a pas d'autres transformations linéaires).*
4. $\mathbf{w}_l \leftarrow \mathbf{w}_{(l+m) \bmod r} \oplus \tilde{A}\mathbf{w}_l$.
5. $l \leftarrow (l+1) \bmod r$, $n \leftarrow n+1$
6. *Aller à l'étape 3.*

On effectue l'opération $\mathbf{y} \leftarrow \tilde{A}\mathbf{w}_n$ par l'algorithme 5.5.3. Pour cet algorithme, α est tel que $p\alpha \equiv 1 \bmod w$, β est tel que $p\beta + q \equiv w - 1 \bmod w$ et γ est tel que $p\gamma + q \equiv 0 \bmod w$.

Algorithme 5.5.3 *Multiplication afin d'obtenir* $\mathbf{y} \leftarrow \tilde{A}\mathbf{w}_n$.

Si $w_n^{(\beta)} = 1$

$$\mathbf{y} \leftarrow ((\mathbf{w}_n \tilde{\gg} \alpha)\ \&\ \mathbf{d}_\gamma) \ \oplus\ \tilde{\mathbf{a}}$$

sinon

$$\mathbf{y} \leftarrow (\mathbf{w}_n \tilde{\gg} \alpha)\ \&\ \mathbf{d}_\gamma$$

Proposition 5.5.2 *Une itération de l'algorithme 5.3.2 suivie d'une permutation* $\tilde{P}^w_{p,q}$ *est équivalente à une itération de l'algorithme 5.5.3.*

Preuve :

La preuve est très similaire à celle de la proposition 5.5.1. Cependant, la fonction π *est telle que* $\pi(i) = pi + q \bmod w$ *et* $\pi^{-1}(j) = p^{-1}j + p^{-1}q \bmod w$.■

5.6 Un exemple d'implantation de GCL polynomial

Dans cette section, on donne un exemple, à la figure 5.1, du code, en langage C, d'un GCL polynomial avec 3 transformations linéaires. Cet exemple est publié dans [20] et porte le nom de `poly96`. Le vecteur **a** pour ce générateur est `dc7348d7 18975f66 2c2ba527`. Les transformations linéaires sont, dans l'ordre, une permutation des coordonnées $P_{23,83}$, un self-tempering $S_{32,10}$ et un tempering de Matsumoto-Kurita où $w = k$, $s = 23$, $t = 47$, $\mathbf{b}$=`2fa51fb4 2e1e2000 03000000` and $\mathbf{c}$ = `78d849e0 55db0000 00000000`.

Afin d'implanter la permutation de coordonnées, on utilise l'algorithme 5.5.1. Les valeurs de α, β et γ pour ce générateurs sont 71, 84 et 59 respectivement et la valeur de $\tilde{\mathbf{a}}$ est `4b24716e fbc6cd96 0ab7ab0c`.

Dans le programme, les variables `state0`, `state1` et `state2` contiennent le vecteur de bits $\mathbf{z}_n$ qui est la valeur de $\mathbf{x}_n$ après permutation. C'est ce vecteur que l'on manipule, au lieu de $\mathbf{x}_n$. C'est l'algorithme 5.5.1 qui permet cela. Les variables qui contiennent le vecteur de bits $\mathbf{y}_n$ sont `y0`, `y1` et `y2`. Lorsque le tempering de Matsumoto-Kurita est exécuté, on obtient la sortie u_n avec `y0 * 2.3283064365e-10`.

Afin que le générateur progresse dans son plus long cycle, il est important que les variables `state0`, `state1` et `state2` ne soient pas toutes nulles. De plus, la sortie a une résolution de 32 bits. Si l'on désire 53 bits de résolution (le maximum admis pour le type `double` en C), il suffit de réactiver l'avant-dernière ligne de code et de rajouter `y1 * 5.421010862247e-20` à la valeur retournée.

Nous avons chronométré la performance de ce générateur et comparé les résultats avec ceux obtenus avec des générateurs connus. L'expérience a été exécutée deux fois : une fois sur un Pentium-III 600MHz et l'autre sur un AMD Athlon 750MHz. Le tableau 5.1 donne les résultats de ces expériences.

Le générateur `poly96*` est le même générateur que `poly96` mais sans aucune transformation linéaire. Ce générateur n'a pas une très bonne équidistribution, comme il est discuté à la section 7.1.1. La description de `MRG32k3a` se trouve dans [16]. Le générateur `MT19937` a été introduit dans [27] et nous avons utilisé l'implantation donnée dans cette publication.

Le générateur `poly96*` est le plus rapide, près de deux fois plus rapide que `poly96`, mais ne peut être recommandé à cause de ses mauvaises propriétés théoriques. Le générateur `poly96` est plus rapide que `MRG32k3a`, mais un peu plus lent que `MT19937`. Le générateur `MT19937` a une période beaucoup plus grande que `poly96`, mais n'est pas ME.

Tableau 5.1 – Comparaison de la vitesse de différents générateurs (temps en secondes pour générer 10^7 nombres)

Générateur	P-III	AMD
MT19937	1.73	1.20
poly96	2.11	1.45
poly96*	0.99	0.76
MRG32k3a	4.84	2.71

```
#define a0  0x4b24716eUL
#define a1  0xfbc6cd96UL
#define a2  0xab7ab0cUL
#define B10 0x2fa51fb4UL
#define B11 0x2e1e2000UL
#define B12 0x03000000UL
#define B20 0x78d849e0UL
#define B21 0x55db0000UL
static unsigned long state0 = 0x1UL, state1 = 0x0UL, state2 = 0x0UL;

double poly96 (void) {
   unsigned long e, y0, y1, y2, w0, w1, w2;

   /* Décalage rotatif de 71 bits vers la gauche */
   w0 = (state0 >> 25) ^ (state2 << 7);
   w1 = (state1 >> 25) ^ (state0 << 7);
   w2 = (state2 >> 25) ^ (state1 << 7);

   /* Élimination du bit 84 et vérification du bit 59 */
   w2 = w2 & 0xfffff7ffUL;
   if (state1 & 0x00000010UL) {
      state0 = w0 ^ a0;  state1 = w1 ^ a1;  state2 = w2 ^ a2;
      }
   else {
      state0 = w0;  state1 = w1;  state2 = w2;
      }

   /* Self-tempering avec c=32 et d=10  */
   e = (state0 ^ state1 ^ state2  ) << 10;
   y0 = state0 ^ e;  y1 = state1 ^ e;  y2 = state2 ^ e;

   /* tempering de M-K avec s=23 et t=47  */
   y0  = y0 ^ (((y1 >> 9) ^ (y0 << 23)) & B10);
   y1  = y1 ^ (((y2 >> 9) ^ (y1 << 23)) & B11);
   y2  = y2 ^ ((y2 << 23) & B12);
   y0  = y0 ^ (((y2 >> 17) ^ (y1 << 15)) & B20);
   /* y1  = y1 ^ ((y2 << 15) & B21); */
   return (y0 * 2.3283064365e-10);
   }
```

Figure 5.1 – Implantation de poly96 en Langage C

Chapitre 6

Le progiciel REGPOLY

Dans ce chapitre nous décrivons le but du progiciel REGPOLY qui a été conçu dans le cadre de cette maîtrise. On parle de ce qui existait avant sa conception et des innovations importantes du progiciel REGPOLY. On retrouve aussi des exemples de recherche de bons générateurs qui démontrent les capacités du progiciel REGPOLY.

6.1 Raison d'être de REGPOLY

Dans le passé, au laboratoire de simulation du département d'informatique et de recherche opérationnelle de l'Université de Montréal, il existait deux logiciels permettant de vérifier l'équidistribution de générateurs à récurrence linéaire modulo 2. L'un était conçu pour les générateurs de Tausworthe combinés basés sur des trinômes sans transformation linéaire à la sortie et l'autre pour les TGFSR combinés avec du tempering de Matsumoto-Kurita à la sortie. Ce deuxième n'était pas très efficace, car son algorithme qui permettait d'optimiser le tempering de Matsumoto-Kurita (qui trouve les masques $\mathbf{b}$ et $\mathbf{c}$) ne fonctionnait pas pour des générateurs avec $L > 7$. Ces programmes utilisaient des procédures semblables, mais qui étaient spécifiques aux générateurs. Il n'y avait donc aucune compatibilité entre le code des deux logiciels.

Ainsi, si l'on désirait vérifier l'équidistribution d'autres types de générateurs, alors on avait deux options : construire des programmes semblables propres à chacun des générateurs ou construire un progiciel qui permet de faire ce que les deux autres permettent de faire, mais qui permet également de vérifier l'équidistribution d'autres générateurs aisément.

Il était devenu évident qu'il fallait construire un progiciel qui réalise la deuxième option étant donné le nombre croissant de ce type de générateurs et des transformations linéaires à appliquer à la sortie de ces générateurs. Le progiciel REGPOLY est donc l'implantation de la deuxième option. Le progiciel REGPOLY répond aux critères de développement suivants :

1. modularité,
2. efficacité, et
3. au moins les mêmes capacités que les deux autres programmes.

En effet, le nouveau progiciel devait être construit de façon modulaire. C'est-à-dire, si l'on décide de rajouter un type de générateur ou de modifier certains aspects du progiciel, alors seulement un nombre minimal de modules s'en trouveraient affectés. Pour atteindre ce but, il faut bien analyser le problème et généraliser le plus possible afin que chaque module soit le plus indépendant possible.

Également, il faut que le progiciel soit aussi efficace, en terme d'effort de calcul, que les deux autres programmes pour les tâches que ceux-ci étaient capables de faire. Ceci peut devenir un obstacle lorsque vient le temps de généraliser les algorithmes.

Finalement, il est évident que l'on ne devait pas construire ce progiciel s'il en résultait une perte de capacités. Ainsi, le progiciel REGPOLY permet d'effectuer au moins les mêmes tâches que les programmes qui existaient auparavant.

D'autres nouvelles fonctionnalités ont été développées dans le progiciel REGPOLY. Voici une liste des celles-ci :

1. Vérifier l'équidistribution de n'importe quel type de générateurs combinés à récurrence linéaire modulo 2 (pas seulement les générateurs de Tausworthe combinés et les TGFSR combinés).

2. Vérifier l'équidistribution des projections ($\tilde{\Delta}_{t_1,\ldots,t_d}$) pour n'importe quel type de générateurs combinés.
3. Vérifier la propriété CF (voir [13] pour plus de détails) si le générateur est ME, pour n'importe quel type de générateurs combinés.
4. Vérifier des générateurs combinés dont chaque composante peut être de type différent.
5. Appliquer des transformations linéaires sur la sortie de chacune des composantes du générateur combiné.
6. Appliquer un algorithme d'optimisation du tempering de Matsumoto-Kurita efficace pour n'importe quel type de générateur combiné.
7. Dans sa version présente, le progiciel implante trois types de transformations linéaires et trois types de générateurs de nombres aléatoires, mais permet d'implanter de nouvelles transformations linéaires ou de nouveaux générateurs facilement à cause du design modulaire du progiciel.
8. Le module qui s'occupe des générateurs de Tausworthe ne permet pas seulement des trinômes, mais aussi des polynômes ayant plus que trois coefficients non nuls.
9. Permettre la création de nouveaux critères de sélections de générateurs combinés.

Le progiciel REGPOLY est en fait un ensemble de facilités, codées à l'avance, qui permettent à quiconque d'écrire des programmes à partir des facilités de REGPOLY. L'ensemble de ces facilités, regroupées en modules, permet à un programmeur de fabriquer des programmes de recherche de générateurs de nombres aléatoires, spécifiques à ses besoins. Le programme `recherche.c`, que l'on peut retrouver dans le guide d'utilisation de REGPOLY (qui se trouve à l'annexe de ce mémoire), essaie de tirer le maximum des capacités présentes du progiciel. C'est avec le programme `recherche.c` que l'on démontre les capacités du progiciel dans la prochaine section.

6.2 Démonstration des capacités de REGPOLY

Dans cette section, on démontre les capacités de REGPOLY par des exemples de recherche. Ces exemples mettent en relief les capacités de REGPOLY comparativement à ses deux ancêtres. Le programme recherche.c a été écrit pour la recherche de générateurs de nombres aléatoires. Il est utilisé pour démontrer les capacités du progiciel REGPOLY dans cette section. Il est aussi utilisé au chapitre 7 pour les recherches de GCL polynomiaux et de TGFSR combinés. On trouve le code de recherche.c dans le guide d'utilisation de REGPOLY. D'autres exemples de recherche se trouvent dans le guide d'utilisation

6.2.1 Exemple 1

Nous sommes intéressés à vérifier l'équidistribution de GCL polynomiaux qui ont des polynômes caractéristiques de degré 32 et auxquels on applique deux transformations linéaires à la sortie : une permutation et un tempering de Matsumoto-Kurita. La résolution du générateur est de $L = 32$. On désire utiliser l'algorithme 4.4.4 d'optimisation du tempering de Matsumoto-Kurita et on recherche des générateurs qui sont ME.

Supposons que l'on permette au programme de vérifier 10 ensembles de paramètres pour les transformations linéaires sur chaque polynôme caractéristique. Un ensemble de paramètres serait, par exemple, la combinaison des paramètres $p = 7$ et $q = 13$ pour la permutation et $\eta = 7$, $\mu = 15$, $\mathbf{b} = (\text{ee12345fe})_{16}$ et $\mathbf{c} = (\text{eec3ca222})_{16}$ (où les vecteurs de bits sont donnés sous la forme hexadécimale). Dans cet exemple, ces combinaisons de paramètres sont choisies au hasard pour p et q, tandis que les paramètres η et μ sont déterminés à l'avance par l'usager. Les vecteurs de bits $\mathbf{b}$ et $\mathbf{c}$ sont choisis par l'algorithme d'optimisation du tempering de Matsumoto-Kurita.

Pour effectuer cette recherche, on appelle le programme avec la commande

```
> recherche exemple1
```

où `exemple1` est le nom du fichier de données principal (figure 6.1). Les polynômes à vérifier sont dans le fichier `32poly.dat` (figure 6.2), tandis que le fichier `trans32.dat` (figure 6.3) indique au programme quelles transformations linéaires utiliser. Pour le format de ces fichiers, il est nécessaire d'aller consulter le guide d'utilisation de REGPOLY donné en annexe de ce mémoire. Également, un échantillonnage de ce que le programme affiche à l'écran est donné aux figures 6.4 et 6.5. Dans la figure 6.4, on affiche un résumé des paramètres de la recherche ainsi que le premier générateur trouvé. Les générateurs de type GCL polynomiaux sont affichés selon deux formats : on affiche le degré des coefficients non nuls du polynôme caractéristique et le vecteur **a** en notation hexadécimale. On affiche également les paramètres des transformations linéaires. Pour la permutation, les valeurs entre parenthèses sont p et q (dans l'ordre) et pour le tempering de Matsumoto-Kurita, η et μ. Les vecteurs **b** et **c** sont donnés en notation hexadécimale.

La figure 6.5 indique que l'on a obtenu 53 générateurs ME en 285 secondes. Les exemples de ce chapitre ont tous été exécutés sur un Pentium-III 600MHz avec RedHat Linux 6.2 comme système d'exploitation. Pour expérimenter avec la performance de l'algorithme 4.4.4, on a effectué des recherches pour exactement le même type de générateurs, mais sans l'algorithme d'optimisation 4.4.4. En 333 secondes de recherche, le progiciel n'a trouvé aucun générateur ME.

Cet exemple démontre que l'algorithme d'optimisation du tempering de Matsumoto-Kurita est efficace, même pour des générateurs avec $L > 7$, et que le tempering de Matsumoto-Kurita n'est pas restreint aux générateurs TGFSR.

```
1                  # 1 composante
-1                 # racine du générateur => horloge
poly 32 32poly.dat # composante 1 : GCL polynomial,L=32, fichier de polynômes : 32poly.dat
1 trans32.dat      #                lin.trans.=oui,fichier de lin.trans.=trans32.dat
10                 # 10 essais de trans.lin. par générateur
0                  # on cherche des générateurs ME
0                  # on ne défini pas de seuil pour écarts en dimension pour résolutions particulières
0                  # pas de vérification de DELTA
```

Figure 6.1 – Fichier de données principal pour l'exemple 1.

```
32  30  25  24  22  15   6   2   0
32  31  30  29  23  21  19  18  16  13  11  10   9   8   7   5   3   1   0
32  28  26  23  21  20  19  18  17  16  13  10   9   7   2   1   0
32  30  29  25  23  19  17  16  15  13  12  11   9   8   7   4   3   1   0
32  31  30  29  28  24  23  21  18  14  13   9   4   3   0
32  31  29  28  24  23  22  21  17  16  15  12  11   9   7   6   5   4   3   1   0
32  31  29  27  26  25  21  20  18  16  15  14  11  10   9   8   7   6   4   3   2   1   0
32  31  28  27  25  24  22  21  17  15  13  12  10   7   6   4   0
32  31  30  24  23  22  21  18  16  14  11  10   9   8   6   5   3   2   0
32  31  26  25  23  22  21  20  17  16  14  12  10   8   5   4   3   2   0
32  31  26  24  19  16  14  12  11  10   8   5   3   2   0
32  28  27  26  24  23  20  18  16  15  14  13  11  10   8   5   4   3   0
32  31  29  28  25  23  22  16  14  11  10   8   6   4   0
```

Figure 6.2 – Fichier `32poly.dat` pour l'exemple 1.

```
2
permut -1 -1
tempMKopt 7 15  -1 0 32
```

Figure 6.3 – Fichier `trans32.dat` pour l'exemple 1.

```
==================================================================
RESUME DE LA RECHERCHE DE GENERATEURS

Racine du générateur des trans. lin. = ( 967680431, 967679431 )

1 composante :
- Composante 1 : Polynomial
  Transformations :
   * Permutation
   * Tempering Matsumoto-Kurita
Nombre d'essais par générateur combiné : 10
Borne sup. pour la somme des écarts dans psi_12 : 0
==================================================================
==================================================================
Degré global :  32

GCL polynomial :
 32 30 25 24 22 15 6 2 0
sous forme hexadécimale :
 43408045
Permutation(11,3)
Tempering Matsumoto-Kurita (7,15)  b = 13ce0a80   c = 55e08000
==================================================================

 ===> GENERATEUR ME


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 6.4 – Résumé des paramètres de recherche et premier générateur trouvé pour l'exemple 1.

```
===========================
   total   =        130

     ME    =         53
   CF-ME   =          2
  retenus  =         53
---------------------------
 CPU (sec) =   285.57
===========================
```

Figure 6.5 – Résumé des résultats de recherche pour l'exemple 1.

6.2.2 Exemple 2

Dans cet exemple, on recherche des générateurs TGFSR à deux composantes avec du tempering de Matsumoto-Kurita appliqué sur chacune des composantes. Les paramètres des TGFSR sont $w_1 = 31$, $r_1 = 3$, $m_1 = 1$ pour la première composante et $w_2 = 29$, $r_2 = 5$, $m_2 = 2$ pour la deuxième composante. Les valeurs de $\mathbf{a}_1$ et $\mathbf{a}_2$ sont indiquées dans les fichiers `93_1.dat` et `145_2.dat`. Plusieurs valeurs de ces vecteurs de bits sont essayées. On recherche des générateurs qui sont tels que $\sum_{\ell \in \Psi_{12}} \Delta_\ell \leq 5$ en utilisant l'algorithme 4.4.4. On essaie 5 ensembles de paramètres pour les transformations linéaires par générateur combiné. Pour tous les générateurs combinés de cet exemple, en utilisant l'équation (1.7), on obtient $L = 29$, puisque $L_1 = w_1 = 31$ et $L_2 = w_2 = 29$.

Le fichier de données principal pour cet exemple est `exemple2` (figure 6.6). Les fichiers contenant les paramètres de chacune des composantes du TGFSR combiné sont `93_1.dat` et `145_2.dat` (figure 6.7). Les fichiers contenant les transformations linéaires à effectuer sur chacune des composantes sont `trans31.dat` et `trans29.dat` (figure 6.8). Le résumé des paramètres de la recherche et le premier générateur affiché sont donnés à la figure 6.9. Le résumé de la recherche est donné à la figure 6.10.

À la figure 6.9, le tableau des écarts Δ_ℓ est affiché puisque le générateur combiné n'est pas ME. On peut voir que $\Delta_\ell \neq 0$ pour $\ell = 17, 18, 19, 24, 25, 26$. On donne aussi la valeur de $\sum_{\ell \in \Psi_{12}} \Delta_\ell$ qui est de 4 pour ce générateur. Cette quantité est représentative de la qualité de l'équidistribution. En général, plus la valeur est près de zéro, plus l'équidistribution est bonne. On donne aussi le degré du polynôme (à la ligne `Degré global`) résultant de la multiplication des polynômes caractéristiques de chacune des composantes. Dans cet exemple, le degré est de 238. Si pgcd$(2^{31\times 3} - 1, 2^{29\times 5})$=1, alors la période maximale du générateur combiné est près de 2^{238}. Enfin, pour chacune des composantes, on donne les paramètres des générateurs et des transformations linéaires appliquées.

À la figure 6.10, on observe que sur les 280 générateurs essayés, 211 sont tels que $\sum_{\ell \in \Psi_{12}} \Delta_\ell \leq 5$. Le temps consacré à vérifier tous ces générateurs a été de 9464 secondes (environ 2h30min).

Cet exemple montre une fois de plus l'efficacité de l'algorithme 4.4.4 sur les générateurs combinés. Malgré le fait qu'aucun des générateurs trouvés ne soit ME, on a observé lors de l'exécution du programme que plusieurs d'entre eux étaient tels que $\sum_{\ell \in \Psi_{12}} \Delta_\ell = 1$.

```
2                   # 2 composantes
-1                  # racine du générateur => horloge
tgfsr 31 93_1.dt    # composante 1 : TGFSR,L=31, fichier de données : 93_1.dt
1 trans31.dat       #                lin.trans.=oui,fichier de lin.trans.= trans31.dat
tgfsr 29 145_2.dt   # composante 1 : TGFSR,L=29, fichier de données : 145_2.dt
1 trans29.dat       #                lin.trans.=oui,fichier de lin.trans.= trans29.dat
5                   # 5 essais de trans.lin. par générateur
5                   # somme des Delta_l pour l dans psi_12 <=5
0                   # on ne définit pas de seuil pour écarts en dimension pour rés. particulières
0                   # pas de vérification de DELTA
```

Figure 6.6 – Fichier de données principal pour l'exemple 2.

```
31 3
a 7 cdae727e f94aba8e c39bde7a c5353d7a e05efeb6 ca35e4ca d0377b06
m 1 1
```

```
29 5
a 8 c1da3168 f966e478 bcf84958 bc5221b8 9d250498 be38a448 c7ff9068 c5637828
m 1 2
```

Figure 6.7 – Fichiers `93_1.dt` et `145_2.dt` pour l'exemple 2.

```
1
tempMKopt 7 15 -1 0 31
```

```
1
tempMKopt 7 14 -1 0 29
```

Figure 6.8 – Fichiers `trans31.dat` et `trans29.dat` pour l'exemple 2.

```
===================================================================
RESUME DE LA RECHERCHE DE GENERATEURS

Racine du générateur des trans. lin. = ( 967680438, 967679438 )

2 composantes :
- Composante 1 : TGFSR
  Transformations :
   * Tempering Matsumoto-Kurita
- Composante 2 : TGFSR
  Transformations :
   * Tempering Matsumoto-Kurita
Nombre d'essais par générateur combiné : 5
Borne sup. pour la somme des écarts dans psi_12 : 5
===================================================================
=================================================================
Degré global : 238

TGFSR :
 w=  31  r=   3  m=   1  a=   cdae727e
Tempering Matsumoto-Kurita (7,15)  b = 38f05400   c = 2a1540e4
-----------------------------------------------------------------
TGFSR :
 w=  29  r=   5  m=   2  a=   c1da3168
Tempering Matsumoto-Kurita (7,14)  b = 4c1cc008   c = 395c33e8
=================================================================

  Tableau des écarts en dimension pour chaque résolution
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
RESOL  |    1|    2|    3|    4|    5|    6|    7|    8|    9|   10|   11|   12|   13|   14|   15|   16|
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
ECART  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
DIM    |  238|  119|   79|   59|   47|   39|   34|   29|   26|   23|   21|   19|   18|   17|   15|   14|
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
RESOL  |   17|   18|   19|   20|   21|   22|   23|   24|   25|   26|   27|   28|   29|
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
ECART  |    1|    1|    1|     |     |     |     |    1|    1|    1|     |     |     |
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
DIM    |   13|   12|   11|   11|   11|   10|   10|    8|    8|    8|    8|    8|    8|
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
--------------------------->SOMMES ECARTS (Psi_12) = 4


+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 6.9 – Résumé des paramètres de recherche et premier générateur trouvé pour l'exemple 2.

```
===========================
   total   =         280

     ME    =           1
   CF-ME   =           0
  retenus  =         211
---------------------------
 CPU (sec) =   9464.76
===========================
```

Figure 6.10 – Résumé des résultats de recherche pour l'exemple 2.

6.2.3 Exemple 3

L'exemple 3 démontre les capacités du progiciel pour calculer le critère $\tilde{\Delta}_{t_1,\ldots,t_d}$. En effet, c'est une variante du programme `recherche` qui a permis aux auteurs de [19] de trouver de bons générateurs pour leurs applications. Ils voulaient de petits générateurs, pour l'intégration quasi-Monte Carlo, qui montraient une bonne équidistribution pour les projections.

On s'inspire ici de leur recherche de petits générateurs de Tausworthe combinés sans transformations linéaires. On recherche des générateurs dont $\tilde{\Delta}_{10,10,10,10}$ est inférieur à 2. On ne met aucune restriction sur la valeur de $\sum_{\ell\in\Psi_{12}} \Delta_\ell$.

Le fichier de données principal pour cet exemple est `exemple3` (figure 6.11) et le fichier contenant les trinômes à vérifier est `trinomes.dat` (figure 6.13). Un échantillon de l'affichage à l'écran est contenu dans la figure 6.14 qui contient le résumé des paramètres de recherche et le premier générateur trouvé. Le résumé des résultats de la recherche est présenté à la figure 6.12.

Dans l'affichage de la figure 6.14, on retrouve les polynômes de chacune des composantes ainsi que leur paramètre s du générateur combiné trouvé. Le premier tableau de cette figure donne la valeur des écarts Δ_ℓ pour chaque bit. Dans le deuxième tableau de cette figure, on donne les écarts maximaux en résolution pour les projections dans chacune des dimensions. Les valeurs de pourcentage qui sont données dans le tableau sont la proportion des projections, pour chacune des dimensions, qui ont un écart en résolution de 0. On donne aussi la valeur du critère $\tilde{\Delta}_{10,10,10,10}$ qui est de 2 pour le générateur montré.

À la figure 6.12, on observe que parmi les 688 générateurs vérifiés, 335 ont été retenus parce qu'ils étaient tels que $\tilde{\Delta}_{10,10,10,10} \leq 2$. Le temps requis pour vérifier tous ces générateurs est moins de 1 seconde.

Cet exemple démontre que le programme recherche.c permet de calculer les écarts de résolution pour les projections (le critère $\tilde{\Delta}_{t_1,\ldots,t_d}$) pour n'importe quel type de générateur, ce que ne permettaient pas les ancêtres de REGPOLY.

```
2                   # 1 composante
-1                  # racine du générateur => horloge
taus 32 trinomes.dat # composante 1 : Tausworthe,L=32, fichier de polynômes : trinomes.dat
0                   #                lin.trans.= non
taus 32 same        # composante 1 : Tausworthe,L=32, fichier de polynômes : trinomes.dat
0                   #                lin.trans.= non
-1                  # on ne vérifie pas l'équidistribution
0                   # on ne définit pas de seuil pour écarts en dimension pour rés. particulières
1 4 10 10 10 10 2   # vérification du critère DELTA(10,10,10,10), doit être <= 2
```

Figure 6.11 – Fichier de données principal pour l'exemple 3.

```
===========================
   total   =        668

     ME    =          0
   CF-ME   =          0
  retenus  =        335
---------------------------
 CPU (sec) =    0.67
===========================
```

Figure 6.12 – Résumé des résultats de recherche pour l'exemple 3.

```
9
3 3 1 0
3 4 1 0
3 5 2 0
3 6 1 0
3 7 1 0
3 7 3 0
3 9 4 0
3 10 3 0
3 11 2 0
```

Figure 6.13 – Fichier `trinomes.dat` pour l'exemple 3.

```
===================================================================
RESUME DE LA RECHERCHE DE GENERATEURS

Racine du générateur des trans. lin. = ( 967680445, 967679445 )

2 composantes :
- Composante 1 : Tausworthe
- Composante 2 : Tausworthe
On vérifie le Critère DELTA( 10, 10, 10, 10)
On conserve quand DELTA(t_1,..,t_s)<=2
===================================================================
=================================================================
Degré global :   7

Générateur de Tausworthe :
 x^3 + x + 1                              s=1
-----------------------------------------------------------------
Générateur de Tausworthe :
 x^4 + x + 1                              s=1
=================================================================

  Tableau des écarts en résolution (dimensions successives)
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
DIM    |    1|    2|    3|    4|    5|    6|    7|    8|    9|   10|
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
ECART  |     |    2|    1|     |     |     |     |     |     |     |
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
RESOL  |    7|    1|    1|    1|    1|    1|    1|    0|    0|    0|
======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+


  Tableau des écarts maximaux en résolution (dimensions non successives)
==========+======+=====+======+
Dimension  |     2|     3|     4|
-----------+------+------+------+
t_i        |    10|    10|     0|
-----------+------+------+------+
ECART      |     2|     1|     0|
-----------+------+------+------+
pourcentage| 77.78| 44.44|100.00|
==========+======+=====+======+
Valeur de DELTA( 10, 10, 10, 10 ) = 2


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 6.14 – Résumé des paramètres de recherche et premier générateur trouvé pour l'exemple 3.

6.2.4 Exemple 4

Dans cet exemple, on montre que le programme recherche.c est capable de vérifier l'équidistribution pour des générateurs combinés dont chacune des composantes est de type différent. Ainsi, on combine un TGFSR avec un générateur de Tausworthe et un GCL polynomial. Sur le TGFSR, on applique un tempering de Matsumoto-Kurita que l'on optimise. Sur le générateur de Tausworthe, on applique un self-tempering. Sur le GCL polynomial, on applique une permutation de coordonnées.

On recherche des générateurs qui sont tels que $\sum_{\ell\in\Psi_{12}} \leq 3$. Le fichier `exemple3` (figure 6.15) est le fichier de données principal pour cet exemple. Les fichiers `trans1.dat`, `trans2.dat` et `trans3.dat` (figure 6.16) contiennent les transformations linéaires à effectuer pour chacune des composantes. La figure 6.18 montre le résumé de la recherche ainsi que le premier générateur trouvé. La figure 6.17 montre le résumé des résultats.

À la figure 6.18, on voit le tableau des écarts Δ_ℓ, la valeur de $\sum_{\ell\in\Psi_{12}}$ et la période du premier générateur trouvé, ainsi que les paramètres de chacune ses composantes. On observe, à la figure 6.17, qu'aucun des 7280 générateurs essayés n'était ME, mais quelques uns d'entre eux, soit 6, répondaient au critère $\sum_{\ell\in\Psi_{12}} \leq 3$. Le temps d'exécution de la recherche est de 309628 secondes, soit un peu plus de 86 heures.

Cet exemple montre que l'on peut combiner des générateurs de différents types et que l'on peut appliquer des transformations linéaires différentes sur chacune des composantes. Avec les programmes disponibles auparavant, il n'était pas possible de vérifier l'équidistribution du générateur formé d'une composante TGFSR et d'une composante Tausworthe. Également, il n'était pas possible d'appliquer des transformations linéaires différentes sur chacune des composantes. En fait, seulement un tempering de Matsumoto-Kurita sur un TGFSR était permis.

```
3 -1                # 3 composantes, racine du générateur => horloge
tgfsr 31 93_1.dt
1 trans1.dat        #                lin.trans.= oui, fichier de trans.lin. : trans1.dat
taus 32 trinomes.dat # composante 2 : Tausworthe, L=32, fichier de polynômes : trinomes.dat
1 trans2.dat        #                lin.trans.= oui, fichier de trans.lin. : trans2.dat
poly 32 32poly.dat  # composante 1 : GCL polynomial,L=32, fichier de polynômes : 32poly.dat
1 trans3.dat        #                lin.trans.= oui, fichier de trans.lin. : trans3.dat
2                   # 2 essais de trans.lin. par générateur
3                   # somme des Delta_l pour l dans psi_12 <= 6
0                   # on ne définit pas de seuil pour écarts en dimension pour rés. particulières
0                   # pas de vérification de DELTA
```

Figure 6.15 – Fichier de données principal pour l'exemple 4.

```
1
tempMKopt 7 15 -1 0 31
```

```
1
selft -1
```

```
1
permut -1 -1
```

Figure 6.16 – Fichier `trans1.dat`, `trans2.dat` et `trans3.dat` pour l'exemple 4.

```
===========================
   total    =       7280

     ME     =          0
   CF-ME    =          0
  retenus   =          6
---------------------------
 CPU (sec) =   309628.01
===========================
```

Figure 6.17 – Résumé des résultats de recherche pour l'exemple 4.

```
====================================================================
RESUME DE LA RECHERCHE DE GENERATEURS

Racine du générateur des trans. lin. = ( 968481500, 968480500 )

3 composantes :
- Composante 1 : TGFSR
  Transformations :
   * Tempering Matsumoto-Kurita
- Composante 2 : Tausworthe
  Transformations :
   * Self-Tempering
- Composante 3 : Polynomial
  Transformations :
   * Permutation
Nombre d'essais par générateur combiné : 2
Borne sup. pour la somme des écarts dans psi_12 : 3
====================================================================
==================================================================
Degré global : 131

TGFSR :
 w=  31  r=   3  m=   1  a=   cdae727e
Tempering Matsumoto-Kurita (7,15)  b = 1d2ea102   c = 75850032
------------------------------------------------------------------
Générateur de Tausworthe :
 x^6 + x + 1                           s=5
Self-Tempering(2)
------------------------------------------------------------------
GCL polynomial :
 32 31 28 27 25 24 22 21 17 15 13 12 10 7 6 4 0
sous forme hexadécimale :
 9b62b4d1
Permutation(19,31)
==================================================================

  Tableau des écarts en dimension pour chaque résolution
=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
RESOL  |    1|    2|    3|    4|    5|    6|    7|    8|    9|   10|   11|   12|   13|   14|   15|   16|
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
ECART  |     |     |     |     |     |     |     |     |     |    1|     |     |    1|     |     |     |
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
DIM    |  131|   65|   43|   32|   26|   21|   18|   16|   14|   12|   11|   10|    9|    9|    8|    8|
=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====
=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
RESOL  |   17|   18|   19|   20|   21|   22|   23|   24|   25|   26|   27|   28|   29|   30|   31|
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
ECART  |     |     |     |     |     |     |     |     |    1|    1|     |     |     |     |     |
-------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
DIM    |    7|    7|    6|    6|    6|    5|    5|    5|    4|    4|    4|    4|    4|    4|    4|
=======+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+=====+
---------------------------->SOMMES ECARTS (Psi_12) = 3


+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 6.18 – Résumé des paramètres de recherche et premier générateur trouvé pour l'exemple 4.

Chapitre 7

Bons générateurs trouvés grâce à REGPOLY

Dans ce chapitre, nous donnons une liste des générateurs qui ont de bonnes équidistributions. Les paramètres de ces générateurs ont été obtenus grâce au progiciel REGPOLY et au programme recherche.c dont le code est donné dans le guide d'utilisation du progiciel et dont on parle au chapitre 6. On donne le résultat des recherches de générateurs GCL simples avec différentes transformations linéaires appliquées, ainsi que des TGFSR combinés avec tempering de Matsumoto-Kurita sur chacune des composantes.

7.1 GCL polynomiaux

Pour la recherche de ces générateurs, on a utilisé pour résolution à la sortie $L = \min(k, 53)$ puisque 53 est le nombre maximum de bits pour la résolution des nombres à virgule flottante selon le standard IEEE 754. On essaie de trouver des GCL polynomiaux, avec polynôme caractéristique de degré 32, 64, 96 et 128, qui ont la meilleure équidistribution possible. Les résultats sont donnés sous forme de tableaux. Le critère qui donne une idée de l'équidistribution est la valeur de $\sum_{\ell \in \Psi_{12}} \Delta_\ell$. Celle-ci est mise dans l'avant-dernière colonne des tableaux. Afficher la valeur de Δ_ℓ pour chaque bit encombrerait les tableaux et ne fournirait pas beaucoup plus d'information. Dans la dernière colonne, on donne une particularité de l'équidistribution de chacun des générateurs. Chacune des sous-sections suivantes traite de la recherche de GCL avec des transformations linéaires différentes. À noter qu'il y a un format de tableau différent pour chacun des types de GCL polynomial et que les vecteurs de bits **a**, **b** et **c** sont représentés sous la forme hexadécimale.

7.1.1 GCL polynomiaux sans transformations linéaires

Ces générateurs ont une très mauvaise équidistribution. En fait, les générateurs ne sont que (1,$\min(k, L)$)-équidistribués ; on obtenait toujours $t_1 = k$ et $t_\ell = 1$ pour $\ell = 2, \ldots, \min(k, L)$ pour tous les polynômes essayés.

7.1.2 GCL polynomiaux avec une permutation de coordonnées

Selon les recherches que nous avons faites, les GCL polynomiaux avec une permutation de coordonnées ne donnent pas de très bonnes équidistributions. Les valeurs de $\Delta_\ell = 0$ pour $\ell > 3$ sont difficiles à obtenir pour des générateurs ayant des polynômes caractéristiques de degrés 32, 64, 96 et 128. Néanmoins, le tableau 7.1 donne une liste des meilleurs générateurs de ce type obtenus. À noter que, dans la première colonne, on retrouve le vecteur de bits **a** propre à chacun des GCL polynomiaux. Dans la deuxième colonne, on retrouve les paramètres p et q de la permutation de coordonnées $P_{p,q}$. Les deux dernières colonnes donnent une idée de l'équidistribution des générateurs.

7.1.3 GCL polynomiaux avec tempering Matsumoto-Kurita

Nous avons effectué des recherches pour des générateurs avec l'algorithme d'optimisation 4.4.4. En comparant les générateurs obtenus avec ceux ayant seulement une permutation de coordonnées, il semble que la permutation soit plus efficace que le tempering de Matsumoto-Kurita afin d'améliorer l'équidistribution. Le tableau 7.2 donne une liste des

Tableau 7.1 – GCL Polynomiaux avec une permutation de coordonnées

Polynôme **a**	Permutation p	 q	$\sum_{\ell\in\Psi_{12}} \Delta_\ell$	Particularités $\Delta_\ell = 0$ pour
Polynômes de degré 128				
`9b7bce2f 42f305b7 ff1149c1 eef10653`	35	37	62	$\ell = 1, 2, 3$
Polynôme de degré 96				
`4acada15 2e647ff5 396caa79`	67	73	44	$\ell = 1, 2, 3$
Polynôme de degré 64				
`246e4912 49a6ad71`	45	45	16	$\ell = 1, 2, 3$
Polynôme de degré 32				
`a6e73761`	7	9	6	$\ell = 1, 2, 3$

meilleurs générateurs obtenus. À remarquer que, dans la première colonne, on retrouve les vecteurs de bits **a** (propres aux GCL polynomiaux) et les vecteurs **b** et **c** du tempering de Matsumoto-Kurita $T_{w,\eta,\mu,\mathbf{b},\mathbf{c}}$. On y retrouve aussi les paramètres η et μ. La valeur de w pour le tempering de Matsumoto-Kurita est k.

7.1.4 GCL polynomiaux avec self-tempering $c = 32$

Parmi les trois transformations linéaires abordées dans ce mémoire, les résultats obtenus semblent démontrer que le self-tempering n'est pas une transformation linéaire très efficace pour améliorer l'équidistribution lorsque celle-ci est utilisée seule. Le tableau 7.3 donne une liste des meilleurs GCL polynomiaux avec self-tempering $S_{32,d}$ obtenus. On retrouve dans la première colonne de ce tableau les valeurs de **a** de chacun des GCL polynomiaux et, dans la deuxième colonne, la valeur du paramètre d du self-tempering $S_{32,d}$.

7.1.5 GCL polynomiaux avec permutations de coordonnées et tempering Matsumoto-Kurita

En appliquant deux transformations linéaires à la sortie du GCL polynomial, on pourrait s'attendre à avoir une meilleure équidistribution que pour les générateurs avec une seule transformation linéaire. C'est en effet ce que démontre le tableau 7.4. Si on compare les valeurs de $\sum_{\ell \in \Psi_{12}} \Delta_\ell$, avec celles des tableaux précédents, on observe qu'il y a une nette amélioration de l'équidistribution. On retrouve même des générateurs qui sont ME pour $k = 32, 64$. Dans la première colonne du tableau, on retrouve les vecteurs $\mathbf{a}$ propres aux GCL polynomiaux ainsi que les vecteurs $\mathbf{b}$ et $\mathbf{c}$ du tempering de Matsumoto-Kurita. On retrouve aussi sur les paramètres η et μ du tempering de Matsumoto-Kurita $T_{w,\eta,\mu,\mathbf{b},\mathbf{c}}$ (où $w = k$) et de la permutation de coordonnées $P_{p,q}$.

7.1.6 GCL polynomiaux avec permutations de coordonnées et self-tempering

En appliquant une permutation de coordonnées et un self-tempering, on obtient des générateurs avec une assez bonne équidistribution. On observe que que la différence d'amélioration de l'équidistribution entre le self-tempering et le tempering de Matsumoto-Kurita quand il n'y a pas d'autres transformations linéaires est plus grande, en général, que lorsqu'une permutation de coordonnées est déjà appliquée. À noter que dans le tableau 7.6, on retrouve, dans l'ordre, le vecteur de bits $\mathbf{a}$ propre à chacun des GCL polynomiaux, les paramètres p et q de la permutation de coordonnées $P_{p,q}$, le paramètre d du self-tempering $S_{32,d}$ et les deux dernières colonnes donnent une idée de l'équidistribution des générateurs.

7.1.7 GCL polynomiaux avec permutations de coordonnées, self-tempering et tempering de Matsumoto-Kurita

En appliquant trois transformations linéaires sur des GCL polynomiaux, il est très facile de trouver des générateurs qui sont ME. Pour les générateurs avec $k = 32, 64, 96$, on obtient beaucoup de générateurs qui sont ME. Tous les générateurs du tableau 7.7 sont ME. Dans la première colonne on retrouve les vecteurs de bits $\mathbf{a}$, $\mathbf{b}$ et $\mathbf{c}$, ainsi que les paramètres p, q, η, μ et d des transformations linéaires $P_{p,q}$, $S_{32,d}$ et $T_{w,\eta,\mu,\mathbf{b},\mathbf{c}}$ où $w = k$.

Tableau 7.2 – GCL Polynomiaux avec tempering de Matsumoto-Kurita

Polynôme/Tempering de Matsumoto-Kurita **a** **b** **c**	η	μ	$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités $\Delta_\ell = 0$ pour
Polynôme de degré 128				

`9b7bce2f 42f305b7 ff1149c1 eef10653`
`428cab5f 00ee7f72 48c02104 40d703e3`
`76781187 3d83207c 00030477 75eb39d4`
	31	63	153	$\ell = 1, 2$

Polynôme de degré 96				

`88b67e28 c697fb20 e0a2b18b`
`2e242a6c 38fe310a 00000000`
`5cad684e 00c38000 00000000`
	23	47	90	$\ell = 1, 2$

Polynôme de degré 64				

`60237e4f 93e6085f`
`54bf7f4a 4a2a0000`
`6db5c2d4 00000000`
	15	31	42	$\ell = 1, 2, 4$

Polynôme de degré 32				

`14bf2687`
`73765501`
`6eac8000`
	7	15	6	$\ell = 1, 2, 4$

Tableau 7.3 – GCL Polynomiaux avec self-tempering $c = 32$

Polynôme **a**	Self-tempering d	$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités $\Delta_\ell = 0$ pour
Polynôme de degré 128			
`8aced852 3c24c43f cd0e910e 8a5db833`	30	255	$\ell = 1$
Polynôme de degré 96			
`d44461dc 0d2dcdd6 62261e1f`	30	168	$\ell = 1$
Polynôme de degré 64			
`877fa931 41669185`	30	90	$\ell = 1$
Polynôme de degré 32			
`e0ad2fab`	17	31	$\ell = 1, 2$

Tableau 7.4 – GCL Polynomiaux avec permutations de coordonnées et tempering de Matsumoto-Kurita

Paramètres			$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités
a	p	q		$\Delta_\ell = 0$ pour
b	s			
c	t			
Polynômes de degré 128				
`d72127f4 496a85a7 d7f7e9d8 d95aca5d`	25	21	26	$\ell = 1, 2, 5$
`6d5e2ab3 51a62440 034e8040 00000000`	31			
`4dbf606d 2684c1dd 00000000 00000000`	63			
`451eaeb8 234b5180 6ae4ce0b 30776841`	89	23	27	$\ell = 1, 3, 5$
`2d0d3957 67a2a184 46800124 00000000`	31			
`77c8a193 168904c5 00000000 00000000`	63			
Polynôme de degré 96				
`82d62790 75d70e40 f71b76f9`	11	31	16	$\ell = 1, 2, 4$
`7e9be354 455a88e6 20000000`	23			
`7ff35820 5f1e0000 00000000`	47			
Polynôme de degré 64				
`877fa931 41669185`	45	43	0	$\ell = 1, \ldots, 53$ (ME)
`77aebcea 38168000`	15			
`5f5ffec5 00000000`	31			
`cba7bc27 13bb667d`	55	47	0	$\ell = 1, \ldots, 53$ (ME)
`6f7ba0ba 0a0a8000`	15			
`5555515d 00000000`	31			
`cba7bc27 13bb667d`	25	55	0	$\ell = 1, \ldots, 53$ (ME)
`244b2fae 11ba0000`	15			
`7cdd0406 00000000`	31			

Tableau 7.5 – GCL Polynomiaux avec permutations de coordonnées et tempering de Matsumoto-Kurita (suite)

Paramètres			$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités
a	p	q		
b	s			
c	t			
Polynôme de degré 32				
`43408045`	32	11	0	ME
`69a4b000`	7			
`7d650000`	15			
`e0ad2fab`	3	29	0	ME
`2c98ca01`	7			
`6d028000`	15			
`14bf2687`	11	9	0	ME
`3e1a8880`	7			
`555c0000`	15			
`f1a46219`	3	13	0	ME
`5e9aa800`	7			
`55678000`	15			

Tableau 7.6 – GCL Polynomiaux avec permutations de coordonnées et self-tempering $c = 32$

Paramètres				$\sum_{\ell\in\Psi_{12}} \Delta_\ell$	Particularités
a	p	q	d		$\Delta_\ell = 0$ pour
Polynôme de degré 128					
`d72127f4 496a85a7 d7f7e9d8 d95aca5d`	81	9	28	26	$\ell = 1, 2, 3$
Polynôme de degré 96					
`b6d91393 832e6a34 b3cd758b`	17	89	25	14	$\ell = 1, 2, 5, 7$
Polynôme de degré 64					
`b5058e44 c394ecab`	27	15	27	8	$\ell = 1, 2, 3, 5, 6, 8, 11$
Polynôme de degré 32					
`c1e54f6d`	7	9	6	1	$\ell = 1, \ldots, 10$

Tableau 7.7 – GCL Polynomiaux avec permutations de coordonnées, self-tempering et tempering de Matsumoto-Kurita

Paramètres			$\sum_{\ell\in\Psi_{12}} \Delta_\ell$	Particularités
a	p	q		
b	η	d		
c	μ			
Polynômes de degré 128				
`6fc343ac af5bfe30 fd3c435d 32a0ca19`	101	67	0	ME
`55ffd469 77f32dfe 2419620a 00000004`	31	18		
`1f0fbf3d 6d164c05 304ab40c 2a581cea`	63			
`74b480cf 73f3a60c 979782a6 787ddc13`	91	97	0	ME
`23d831ef 295f73be 061a1808 00000001`	31	22		
`07edeca6 5a92f304 2e241c80 31a06893`	63			
`74b480cf 73f3a60c 979782a6 787ddc13`	15	91	0	ME
`4b753dc1 5f7f5ce0 3200c412 00000003`	31	20		
`7d5a6edd 43c7c76c 26fcbc2c 6d66d405`	63			

Tableau 7.8 – GCL Polynomiaux avec permutations de coordonnées, self-tempering et tempering de Matsumoto-Kurita (suite)

Paramètres			$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités
a	p	q		
b	η	d		
c	μ			
Polynômes de degré 96				
dc7348d7 18975f66 2c2ba527	23	8	0	ME
2fa51fb4 2e1e2000 03000000	23	10		
78d849e0 55db0000 00000000	47			
dc7348d7 18975f66 2c2ba527	79	89	0	ME
2ff0f509 15e22881 00800001	23	13		
1772e059 22c58000 00000000	47			
3bd2c03c cdeb771a 274a8575	83	13	0	ME
555b2a29 77029449 06800000	23	19		
4aa5abe6 089a0000 00000000	47			
c5626c3e 0b401252 98b1ef2f	79	59	0	ME
779a510d 77a2284a 12800001	23	11		
5caf8df0 57ed8000 00000000	47			
Polynômes de degré 64				
22440fb5 31d44c91	19	15	0	ME
69fc91ec 0a900000	15	29		
0fcc26ce 00000000	31			
72c4b2f3 39055cfb	25	5	0	ME
5e75187c 008a8000	15	20		
26675cf3 00000000	31			
4549aec8 e233e0e9	53	43	0	ME
76371a02 40418000	15	23		
7926dc20 00000000	31			

Tableau 7.9 – GCL Polynomiaux avec permutations de coordonnées, self-tempering et tempering de Matsumoto-Kurita (suite)

Paramètres			$\sum_{\ell \in \Psi_{12}} \Delta_\ell$	Particularités
a	p	q		
b	η	d		
c	μ			
Polynômes de degré 32				
`43408045`	5	9	0	ME
`5fa62080`	7	5		
`58598000`	15			
`628bbb9b`	21	29	0	ME
`273a3100`	7	24		
`389c8000`	15			
`f1a46219`	19	19	0	ME
`6bb04880`	7	2		
`65538000`	15			
`b1e39afb`	23	17	0	ME
`76860400`	7	20		
`03fd8000`	15			

7.2 TGFSR combinés à deux et à trois composantes

Dans cette section, on recherche des générateurs TGFSR combinés sur lesquels on applique un tempering de Matsumoto-Kurita sur chacune des composantes. L'algorithme 4.4.4 est utilisé afin de trouver les valeurs de $\mathbf{b_j}$ et $\mathbf{c_j}$ du tempering de Matsumoto-Kurita de chacune des composantes.

Lorsqu'on désire combiner des générateurs, il est important que le plus grand commun diviseur (pgcd) des périodes de chacune des composantes soit 1. La période de la composante j est $2^{w_j r_j} - 1$. Un théorème connu de la théorie des nombres (voir [28]) dit que si le pgcd de k_1 et k_2 est différent de 1, alors il en sera de même pour le pgcd de 2^{k_1} et 2^{k_2}. Il faut donc trouver des composantes telles que $w_j r_j$ est premier par rapport à $w_i r_i$ où i est l'indice d'une autre composante.

Cette constatation restreint les combinaisons de générateurs que l'on peut faire si on désire avoir des générateurs qui ont une période maximale. Ainsi, on ne pourra avoir deux composantes avec la même valeur de w_j ou r_j et les w_j doivent être premiers par rapport aux r_i $(i \neq j)$. Par exemple, pour obtenir un générateur une petite période (utilisé pour l'intégration numérique avec les méthodes quasi-Monte Carlo), avec des valeurs de w_j près de 32, on obtient $w_1 = 32$, $r_1 = 2$, $w_2 = 31$ et $r_2 = 3$. Ceci produit un générateur combiné qui a une période approximative de 2^{157}. Pour obtenir des générateurs combinés avec des périodes plus courtes, il faut choisir des valeurs de w_j plus petites. Dans le cas des TGFSR, on a $L_j = w_j$. Puisque l'on désire avoir une résolution d'au moins 32 bits, alors choisir des valeurs de w_j plus petites n'est nullement désiré. On désire au moins avoir une résolution de 32 bits quand c'est possible. On pourrait prendre $L_j > w_j$, mais l'implantation serait moins rapide. Par exemple, on pourrait faire cela en prenant, pour chacune des composantes, $\mathbf{y}_{n,j} = \text{trunc}_{L_j} \mathbf{z}_{n,j} | \mathbf{z}_{n-1,j}$. Cette stratégie n'est pas utilisée dans les résultats, mais illustre qu'il est possible de contourner cet obstacle.

Les deux prochaines sous-sections donnent le résultat de recherches de générateurs combinés à deux et trois composantes. Matsumoto et Kurita [26] ont montré que, pour les TGFSR, il existe une borne $\tilde{t}_\ell$ strictement inférieure à t^*_ℓ pour certaines valeurs de ℓ (voir section 5.3). On savait donc qu'un TGFSR simple ne pouvait être ME. Mais, on ne savait pas si un TGFSR combiné pouvait être ME. Les résultats démontrent que cela est possible puisqu'on a trouvé plusieurs TGFSR qui étaient ME.

7.2.1 TGFSR à deux composantes

Dans le tableau 7.10, on retrouve une liste de TGFSR à deux composantes sur lesquelles on applique un tempering de Matsumoto-Kurita sur chacune des composantes. Ces générateurs sont très bien équidistribués et quelques-uns sont ME.

Dans la première colonne, on retrouve la valeur de k du générateur combiné. Dans les quatre premières colonnes, on retrouve les paramètres $\mathbf{a}$, w, r et m de chacune des composantes et, dans les quatre colonnes suivantes, il y a les paramètres η, μ, $\mathbf{b}$ et $\mathbf{c}$ du tempering de Matsumoto-Kurita appliqué sur chacune des composantes. La dernière colonne affiche les valeurs de ℓ pour lesquelles l'équidistribution ne démontre pas $\Delta_\ell = 0$. Les chiffres entre parenthèses indiquent la valeur de Δ_ℓ. S'il n'y a pas de chiffres entre parenthèses, alors la valeur de Δ_ℓ est de 1.

7.2.2 TGFSR à trois composantes

Dans cette sous-section, on effectue le même type de recherche que dans la sous-section précédente, mais pour des TGFSR combinés à trois composantes sur lesquelles on effectue un tempering de Matsumoto-Kurita. Le format du tableau 7.13 est le même que le tableau 7.10 sauf que la dernière colonne est enlevée puisque tous les générateurs du tableau sont ME.

Tableau 7.10 – TGFSR Combinés à 2 composantes

k	**a**	w	r	m	η	μ	**b**	**c**	$\Delta_\ell \neq 0$
147	f999d3dc	30	2	1	7	14	29e49000	6f3b2d1c	18,24
	c7ba8718	29	3	1	7	14	43944400	798a1520	
147	de11b16c	30	2	1	7	14	72501004	17b52998	18,24
	cba88ff8	29	3	1	7	14	4dec5800	57c60398	
157	ebccfa59	32	2	1	7	15	55fed200	6d608023	26
	f887875e	31	3	2	7	15	52c68402	4ead0014	
157	ebccfa59	32	2	1	7	15	7cbce401	72298029	26
	d68127ce	31	3	2	7	15	4db61100	7ccd003e	
238	cdae727e	31	3	1	7	15	2a5a0902	1ccf0006	ME
	bc5221b8	29	5	2	7	14	59d44000	33ca0000	
238	cdae727e	31	3	1	7	15	70ca6100	38980004	ME
	c7ff9068	29	5	2	7	14	47aca800	3ac60000	
238	f94aba8e	31	3	1	7	15	3664a500	53ac0000	ME
	f966e478	29	5	2	7	14	5a584000	3d4a0000	
245	f94aba8e	31	5	3	7	15	66ae4602	3753001a	27
	8ec4aa8c	30	3	2	7	14	20f02200	4bab0000	
245	f94aba8e	31	5	3	7	15	71468a00	67d2000c	27
	ba56f43c	30	3	2	7	14	43805004	56a60008	
251	9965523b	32	3	1	7	15	4dc60600	23d30014	27
	f45c111a	31	5	2	7	15	5594e302	79880014	
251	9965523b	32	3	1	7	15	05ec8680	63578001	27
	f45c111a	31	5	2	7	15	6636a502	5eee000a	
251	9965523b	32	3	1	7	15	3c2aa101	7ef3800c	27
	ba693616	31	5	2	7	15	52e25502	3b930008	
251	9965523b	32	3	1	7	15	7d9a5281	6f4a0015	27
	9b6bf432	31	5	2	7	15	34da2300	599e0016	

Tableau 7.11 – TGFSR Combinés à 2 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**	$\Delta_\ell \neq 0$
307	a1d5ac42	31	7	2	7	15	6e568900	5cbc006a	25,26,27
	ab2a19bc	30	3	1	7	14	7bd4e804	7ae9005c	
307	a1d5ac42	31	7	2	7	15	373c8900	55d40058	25,26,27
	8fce15bc	30	3	1	7	14	1abc5000	745d000c	
313	b3de2e15	32	3	2	7	15	32dc9800	6eac8033	24,26,28
	a1d5ac42	31	7	2	7	15	13612900	6a5d00a4	
313	b3de2e15	32	3	1	7	15	42ce9081	5edf801c	24,26,28
	a1d5ac42	31	5	2	7	15	1ced2902	7a5280cc	
358	f94aba8e	31	5	3	7	15	77369502	7b970022	25,27
	bcf84958	29	7	4	7	14	55088808	66ac0008	
358	f94aba8e	31	5	3	7	15	2e9c1000	2b4a0040	25,26,27
	bcf84958	29	7	4	7	14	1da93008	5e3a0060	
377	b07c9b61	32	5	1	7	15	48565300	6f8e800c	28,29
	a6712136	31	7	3	6	14	45d55a00	5ef58008	
377	b07c9b61	32	5	1	7	15	337e2201	17588009	28,29
	a6712136	31	7	3	6	14	57a44882	5a6b0002	
536	af15a66e	31	7	3	6	14	36a91100	445a0008	28
	b3e2fa08	29	11	2	7	14	73506400	5da40008	
536	af15a66e	31	7	3	6	14	6cb8c702	31f90002	28
	9e97eac8	29	11	2	7	14	66bc8808	6b980000	
536	af15a66e	31	7	3	6	14	2edc6682	1fb48002	28
	e3107e48	29	11	2	7	14	4de5c008	7b360008	
547	f15a885e	31	7	3	6	14	31c09100	265e0020	26,28
	cc4d219c	30	11	4	7	14	66884800	37510014	
547	a6712136	31	7	3	6	14	6d991900	4743003a	26,27,28
	cc4d219c	30	11	4	7	14	46356804	7b610024	

Tableau 7.12 – TGFSR Combinés à 2 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**	$\Delta_\ell \neq 0$
607	a1d5ac42	31	7	2	6	14	11606202	388e80ee	24,25,26,28
	fdfbc1dc	30	13	7	7	14	6668a204	3aaa007c	
607	f4e1ce72	31	7	2	6	14	03994082	592300b6	24,25,26,28
	fdfbc1dc	30	13	7	7	14	6320c804	7aca00c4	
718	9b6bf432	31	11	2	6	14	0568a302	6bf50008	28
	9a911d68	29	13	2	7	14	21452808	4e2a0000	
718	9b6bf432	31	11	2	6	14	3cb8b980	2ef7800a	27(2),28
	8f11df88	29	13	2	7	14	6ed89c08	37d20010	
731	9b6bf432	31	11	2	6	14	03f42500	3d6a000a	27,28(2),29
	88171a7c	30	13	12	7	14	6d4c8400	79370008	
755	8e0c66a5	32	11	7	7	15	26cc5a01	2f4e8006	29(2),30
	adf8dbc6	31	13	2	6	14	577c8d00	33548004	
755	8e0c66a5	32	11	7	7	15	351a0c00	4e990005	29(2),30
	a1d5ac42	31	13	2	6	14	175c4882	563e8006	
943	d84be803	32	13	7	7	15	67ec0501	5eb580cd	23,29(2),30
	e0e599be	31	17	4	6	14	5b908902	33d38156	

Tableau 7.13 – TGFSR Combinés à 3 composantes

k	**a**	w	r	m	η	μ	**b**	**c**
454	9965523b	32	3	1	7	15	36740900	2d818002
	f45c111a	31	5	2	7	15	1aa0c400	773e0002
	a6250bb8	29	7	4	7	14	39b45800	16620000
454	9965523b	32	3	1	7	15	1d502401	52420005
	f45c111a	31	5	2	7	15	00f44802	345b0004
	a6250bb8	29	7	4	7	14	53d40800	17c00000
454	9965523b	32	3	1	7	15	03ae4d00	3ff38007
	f45c111a	31	5	2	7	15	250eb100	59e40000
	a6250bb8	29	7	4	7	14	434c5000	1efc0000
456	b07c9b61	32	5	1	7	15	49d28980	5fca8005
	cdae727e	31	3	1	7	15	709e0c00	566f0002
	e550de08	29	7	6	7	14	25c87000	5c940000
456	b07c9b61	32	5	1	7	15	15ea0a00	17e98003
	cdae727e	31	3	1	7	15	7bf26400	3f460004
	e550de08	29	7	6	7	14	1b4c8000	316c0000
456	b07c9b61	32	5	1	7	15	23329101	6abe8003
	cdae727e	31	3	1	7	15	61527000	384f0000
	888f8348	29	7	6	7	14	166ca000	38040000
458	b3de2e15	32	3	2	7	15	71fe0800	6c278003
	a1d5ac42	31	7	2	7	15	358e6500	36a80000
	c1da3168	29	5	2	7	14	0a38b000	2c220000
458	b3de2e15	32	3	2	7	15	3a3a2001	3a6f0006
	a1d5ac42	31	7	2	7	15	375a0602	5f3e0000
	c1da3168	29	5	2	7	14	2f60c008	75aa0000
458	b3de2e15	32	3	2	7	15	35020101	10dd0006
	a1d5ac42	31	7	2	7	15	17de0a00	4d440000
	c1da3168	29	5	2	7	14	66b48000	32380000

Tableau 7.14 – TGFSR Combinés à 3 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**
462	acd23ee3	32	70	3	7	15	5ad66801	37370004
	ced60356	31	3	2	7	15	773ada00	6feb0006
	e3562988	29	5	3	7	14	7c713800	4f220000
462	acd23ee3	32	70	3	7	15	77da4d01	369b8003
	ced60356	31	3	2	7	15	58060902	0cda0002
	e3562988	29	5	3	7	14	6b844000	506c0000
462	acd23ee3	32	70	3	7	15	23540801	3e4d8007
	ced60356	31	3	2	7	15	52622100	50930006
	eb12f468	29	5	3	7	14	459c4008	52f60000
464	c1283985	32	5	4	7	15	55a69101	68f38002
	af15a66e	31	7	3	7	15	2f002002	14a60006
	c7ba8718	29	3	1	7	14	00c04808	5b060000
464	c1283985	32	5	4	7	15	3f262400	53138002
	af15a66e	31	7	3	7	15	3f4e4802	276d0000
	c2eaf5d8	29	3	1	7	14	3af14408	514a0000
466	cfae8af3	32	7	6	7	15	26ba6501	3a818006
	f94aba8e	31	5	3	7	15	19382200	73b60000
	fea4abc8	29	3	2	7	14	02541800	16540000
466	cfae8af3	32	7	6	7	15	09ba8b00	67e60004
	f94aba8e	31	5	3	7	15	2c8a0402	1a7c0002
	fea4abc8	29	3	2	7	14	64204000	14100000
466	cfae8af3	32	7	6	7	15	67c21a01	2d648003
	f94aba8e	31	5	3	7	15	1d44c002	68530004
	e915deb8	29	3	2	7	14	2368c000	774e0000
1224	e0ee6d22	31	11	9	7	15	449e1200	29dc0006
	88171a7c	30	13	12	7	14	31980000	11690000
	b113e2a8	29	17	3	7	14	5e190800	432a0000

Tableau 7.15 – TGFSR Combinés à 3 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**
1224	e0ee6d22	31	11	9	7	15	4b568a00	750c0000
	88171a7c	30	13	12	7	14	6c400404	4fb10004
	b113e2a8	29	17	3	7	14	77745000	7cdc0000
1224	e0ee6d22	31	11	9	7	15	1e3e4000	252e0006
	88171a7c	30	13	12	7	14	77ac0000	713c0000
	b113e2a8	29	17	3	7	14	38611800	561a0000
1226	ce4e3616	31	13	11	7	15	39a80202	593c0004
	cc4d219c	30	11	4	7	14	75041004	0c9a0004
	bdfee2f8	29	17	13	7	14	56882c00	2be40000
1226	ce4e3616	31	13	11	7	15	0b2c2002	125b0006
	cc4d219c	30	11	4	7	14	2a9c2400	5d4e0004
	bb792438	29	17	13	7	14	3f410800	62e80000
1226	ce4e3616	31	13	11	7	15	1d100000	11de0000
	cc4d219c	30	11	4	7	14	31e12200	6be80004
	bb792438	29	17	13	7	14	5d302400	2d4e0000
1234	e0e599be	31	17	4	7	15	152e3300	7d810002
	bb64c4d4	30	11	8	7	14	13d08c00	3b2a0000
	8f11df88	29	13	2	7	14	19788000	64440000
1234	e0e599be	31	17	4	7	15	4fb25402	3adf0000
	bb64c4d4	30	11	8	7	14	54642804	0b410000
	8f11df88	29	13	2	7	14	39004008	780a0000
1234	e0e599be	31	17	4	7	15	40940500	269e0006
	bb64c4d4	30	11	8	7	14	509c8800	62180000
	8f11df88	29	13	2	7	14	34c50408	41d60000
1236	a701b236	31	17	14	7	15	08561102	28ab0000
	fdfbc1dc	30	13	7	7	14	28d91000	4e540000
	e4bfe8f8	29	11	9	7	14	41bd1000	45a00000

Tableau 7.16 – TGFSR Combinés à 3 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**
1236	a701b236	31	17	14	7	15	571e1100	6dca0000
	fdfbc1dc	30	13	7	7	14	4a9c0004	04b10000
	e4bfe8f8	29	11	9	7	14	42214000	585c0000
1236	a701b236	31	17	14	7	15	63b48102	62ad0006
	fdfbc1dc	30	13	7	7	14	36e88600	3be00000
	e4bfe8f8	29	11	9	7	14	35380000	032c0000
1248	8e0c66a5	32	11	7	7	15	12868201	5b390003
	a51a1c6a	31	13	2	7	15	6fa63602	3b750006
	b113e2a8	29	17	3	7	14	2354d400	35f20000
1248	8e0c66a5	32	11	7	7	15	552c0000	42a58006
	a51a1c6a	31	13	2	7	15	33d69500	7bc10004
	f711c598	29	17	3	7	14	67395800	76b20000
1248	8e0c66a5	32	11	7	7	15	403a0200	51900002
	a51a1c6a	31	13	2	7	15	536c1102	19c70004
	8b10ed68	29	17	3	7	14	2d850000	4e5c0000
1250	d84be803	32	13	7	7	15	26a68400	432a8000
	9b6bf432	31	11	2	7	15	5c941200	194f0006
	bdfee2f8	29	17	13	7	14	50280008	2aaa0000
1250	cae92dfd	32	13	7	7	15	32fec001	62768005
	9b6bf432	31	11	2	7	15	3756c000	634d0004
	bdfee2f8	29	17	13	7	14	25a45400	17280000
1250	cae92dfd	32	13	7	7	15	5c480901	6fc10007
	9b6bf432	31	11	2	7	15	73842202	17e80000
	bdfee2f8	29	17	13	7	14	22f56400	79ba0000
1256	8c523a2b	32	11	9	7	15	79121981	0ef68004
	e0e599be	31	17	4	7	15	496a1602	1f430004
	8f11df88	29	13	2	7	14	08cca800	7a7e0000

Tableau 7.17 – TGFSR Combinés à 3 composantes (suite)

k	**a**	w	r	m	η	μ	**b**	**c**
1256	8c523a2b	32	11	9	7	15	080c0c80	16db0000
	e0e599be	31	17	4	7	15	1c884000	26690006
	8f11df88	29	13	2	7	14	6a104400	55d20000
1256	8c523a2b	32	11	9	7	15	19ee0400	72598004
	e0e599be	31	17	4	7	15	4a440800	1c740000
	8f11df88	29	13	2	7	14	3a295400	55bc0000
1262	dda697c1	32	13	12	7	15	794ab281	5f5e0002
	a701b236	31	17	14	7	15	5bd84900	2d9b0002
	e4bfe8f8	29	11	9	7	14	75a10000	71c40000
1262	dda697c1	32	13	12	7	15	43a43000	39ed8000
	a701b236	31	17	14	7	15	1a7e8100	4cac0004
	e4bfe8f8	29	11	9	7	14	06e50400	458c0000
1262	dda697c1	32	13	12	7	15	45480401	23fa0001
	a701b236	31	17	14	7	15	261e0900	14ad0006
	f732adc8	29	11	9	7	14	0bf12008	48f20000
1266	fabcaffd	32	17	13	7	15	5e385500	2baf8001
	ce4e3616	31	13	11	7	15	7570d602	7fa50000
	b3e2fa08	29	11	2	7	14	24918400	71500000
1266	fabcaffd	32	17	13	7	15	2ace3201	396c8005
	ce4e3616	31	13	11	7	15	444cb400	7ac20000
	b3e2fa08	29	11	2	7	14	40c40400	13ce0000
1266	fabcaffd	32	17	13	7	15	15861600	3bb30007
	ce4e3616	31	13	11	7	15	1c760000	12460006
	b3e2fa08	29	11	2	7	14	11b8c008	72160000

Chapitre 8

Conclusion

La principale contribution de ce mémoire est le progiciel REGPOLY qui sera disponible au laboratoire de simulation du Département d'informatique et de recherche opérationnelle de l'Université de Montréal. Pour les besoins de ce mémoire, un cadre général pour définir et analyser les générateurs à récurrence linéaire modulo 2 a été développé et est utilisé dans le progiciel REGPOLY. Ce cadre permet d'appliquer des transformations linéaires à la sortie de tous les générateurs de ce type.

Voici un résumé des nouveautés de ce mémoire qui sont implantés dans REGPOLY :

- Le progiciel implante des transformations linéaires nouvelles dont la permutation de coordonnées et le self-tempering.
- Il implante aussi un algorithme permettant d'optimiser le tempering de Matsumoto-Kurita pour les générateurs combinés.
- Le progiciel implante une généralisation de l'algorithme QuickTaus.

Le progiciel a été conçu de façon à ce qu'il soit facile de rajouter des capacités grâce à sa structure modulaire. Un des ajouts à implanter dans un avenir prochain est la capacité de vérifier différents critères sur des générateurs Mersenne Twister [27]. Aussi, il pourrait y avoir des améliorations à apporter aux algorithmes qui vérifient l'équidistribution. En effet, pour vérifier l'équidistribution, il faut diagonaliser plusieurs matrices qui ont plusieurs colonnes en commun, ce qui consomme la majorité des efforts de calculs lors d'une recherche de bons générateurs. Il serait intéressant de chercher des algorithmes permettant de "récupérer" le travail fait pour diagonaliser $B_{\lfloor k/\ell \rfloor, \ell}$ pour ensuite diagonaliser, par exemple, $B_{\lfloor k/(\ell+1) \rfloor, (\ell+1)}$. Cet aspect n'a pas été abordé dans ce mémoire étant donné la complexité du problème. Il serait aussi intéressant d'implanter la nouvelle méthode permettant de vérifier l'équidistribution qui est expliquée dans [3]. Cette méthode pourrait permettre de vérifier l'équidistribution plus rapidement quand le degré du générateur combiné est grand (par exemple, $k \geq 2^{2000}$).

En plus du progiciel REGPOLY, il y a deux nouvelles techniques qui améliorent l'implantation de GCL polynomiaux et de TGFSR lorsqu'on applique une permutation des coordonnées à la sortie de ceux-ci. Ces techniques permettent de faire évoluer la récurrence d'un générateur dans l'espace des vecteurs $\mathbf{z}_n$ plutôt que dans l'espace des états $\mathbf{x}_n$.

Auparavant, on savait qu'il était impossible de trouver un TGFSR qui soit ME. Par contre, on ne savait pas s'il existait des TGFSR combinés qui soient ME. On a démontré qu'il en existe puisqu'on en a trouvé.

La composition de transformations linéaires semble donner une bonne équidistribution aux GCL polynomiaux. Le problème avec la composition de transformations linéaires est le temps nécessaire pour les effectuer. Il serait intéressant de trouver de nouvelles transformations linéaires qui permettent une aussi bonne équidistribution que la composition de transformations linéaires, mais avec des temps d'exécution beaucoup plus rapides.

ANNEXE

Guide d'utilisation de REGPOLY

FIN DE L'ANNEXE

Guide d'utilisation de REGPOLY

Bibliographie

[1] L. Blum, M. Blum, and M. Schub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2) :364–383, 1986.

[2] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, second edition, 1987.

[3] R. Couture and P. L'Ecuyer. Lattice computations for random numbers. *Mathematics of Computation*, 69(230) :757–765, 2000.

[4] R. Couture, P. L'Ecuyer, and C. Lemieux. Polynomial lattice rules. en préparation, 1999.

[5] E. J. Dudewicz and T. G. Ralley. *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code*. American Sciences Press, Columbus, Ohio, 1981.

[6] D. E. Knuth. *The Art of Computer Programming, Vol. 2*. Addison-Wesley, Reading, Mass., deuxième edition, 1981.

[7] H. Krawczyk. How to predict congruential generators. In G. Brassard, editor, *Advances in Cryptology : Proceedings of CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 138–153. Springer-Verlag, Berlin, 1990.

[8] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, third edition, 2000.

[9] P. L'Ecuyer. Efficient and portable combined random number generators. *Communications of the ACM*, 31(6) :742–749 and 774, 1988. See also the correspondence in the same journal, 32, 8 (1989) 1019–1024.

[10] P. L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10) :85–97, 1990.

[11] P. L'Ecuyer. Testing random number generators. In *Proceedings of the 1992 Winter Simulation Conference*, pages 305–313. IEEE Press, déc 1992.

[12] P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53 :77–120, 1994.

[13] P. L'Ecuyer. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation*, 65(213) :203–213, 1996.

[14] P. L'Ecuyer. Random number generation. In Jerry Banks, editor, *Handbook of Simulation*, pages 93–137. Wiley, 1998.

[15] P. L'Ecuyer. Random number generators and empirical tests. In P. Hellekalek, G. Larcher, H. Niederreiter, and P. Zinterhof, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 127 of *Lecture Notes in Statistics*, pages 124–138. Springer, New York, 1998.

[16] P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1) :159–164, 1999.

[17] P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation*, 68(225) :261–269, 1999.

[18] P. L'Ecuyer and P. Hellekalek. Random number generators : Selection criteria and testing. In P. Hellekalek and G. Larcher, editors, *Random and Quasi-Random Point Sets*, volume 138 of *Lecture Notes in Statistics*, pages 223–265. Springer, New York, 1998.

[19] P. L'Ecuyer and C. Lemieux. Quasi-monte carlo via linear shift-register sequences. In *Proceedings of the 1999 Winter Simulation Conference*, pages 336–343. IEEE Press, 1999.

[20] P. L'Ecuyer and F. Panneton. A new class of linear feedback shift register generators. In *Proceedings of the 2000 Winter Simulation Conference*, Pistacaway, NJ, déc 2000. IEEE Press. Parution prochaine.

[21] P. L'Ecuyer and R. Proulx. About polynomial-time "unpredictable" generators. In *Proceedings of the 1989 Winter Simulation Conference*, pages 467–476. IEEE Press, déc 1989.

[22] C. Lemieux. *L'utilisation de règles de réseau en simulation comme technique de réduction de la variance.* PhD thesis, Université de Montréal, mai 2000.

[23] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, Cambridge, 1986.

[24] G. Marsaglia, B. Narasimhan, and A. Zaman. A random number generator for PC's. *Computer Physics Communications*, 60 :345–349, 1990.

[25] M. Matsumoto and Y. Kurita. Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation*, 2(3) :179–194, 1992.

[26] M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation*, 4(3) :254–266, 1994.

[27] M. Matsumoto and T. Nishimura. Mersenne twister : A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1) :3–30, 1998.

[28] Robert J. McEliece. *Finite Fields for Computer Scientists and Engineers*, volume 23 of *The Kluwer International Series in Engineering and Computer Science.* Kluwer Academic Publishers, Boston, 1986.

[29] J. B. Plumstead. Inferring a sequence generated by a linear congruence. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 153–159, 1982.

[30] R. C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19 :201–209, 1965.

[31] S. Tezuka. *Uniform Random Numbers : Theory and Practice.* Kluwer Academic Publishers, Norwell, Mass., 1995.

[32] J. P. R. Tootill, W. D. Robinson, and D. J. Eagle. An asymptotically random Tausworthe sequence. *Journal of the ACM*, 20 :469–481, 1973.