Université de Montréal

Accélération de prédiction génétique par implémentation hautement parallèle sur un matériel re-configurable

Par

Mohamed Toufik Zerarka

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

Août, 2004

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Accélération de prédiction génétique par implémentation hautement parallèle sur un matériel re-configurable

Présenté par :

Mohamed Toufik Zerarka

A été évalué par un jury composé des personnes suivantes :

Miklós csürös
président rapporteur

El mostapha aboulhamid
directeur de recherche

Jean-pierre david
codirecteur

Petko valtchev
membre de jury

Mémoire accepté le 11 novembre 2004

Résumé

Un des défis de la génétique moderne consiste à modéliser l'évolution temporelle de l'expression de certains gènes dans un système donné. Découvrir les points stables du système (les attracteurs) est un aspect important pour la recherche mais il exige des ressources de calcul très importantes. Dans ce mémoire, nous abordons ce problème par une approche de conception conjointe matériel-logiciel en utilisant une plateforme réunissant un processeur ARM pour la partie logicielle et un FPGA de taille moyenne pour la partie matérielle. Cette solution permet la manipulation de centaines de gènes, tandis qu'une solution purement logicielle présentée récemment dans la littérature ne permettait pas de manipuler plus que 15 gènes en raison des contraintes mémoire. En termes de vitesse de calcul, le FPGA nous a permis d'obtenir un gain de quatre ordres de grandeur comparativement à la solution purement logicielle.

Mots clés:

Émulation FPGA, réseaux de régulation génétique, réseaux probabilistes booléens, attracteurs, conception conjointe matériel-logiciel.

Abstract

One of the challenges of the modern Genetics consists in modelling the temporal evolution of the expression of certain genes in a given system. To discover the stable points of the system (attractors) is an aspect important for the research but it requires a very computing intensive and memory demanding task. We show a very efficient approach with impressive results to deal with this task using a platform containing an ARM and a medium size FPGA. While a pure software solution could not handle more than 15 genes due to memory constraints, The FPGA solution could handle hundreds of genes. Gains of three orders of magnitude have been reached compared to a pure software approach.

Keywords : FPGA emulation, Gene Regulatory Networks, Probabilistic Boolean Networks, attractors, Hardware/software codesign.

Table de matières

Listes des figures

Liste des tables

Liste d'exemple de codes

Liste des abréviations

PBN : Probabilistic Boolean Networks

FPGA : Field Programmable Gate Array

LFSR : Linear Feedback Shift Register

LUT : Look up Table

IOB : Input Output Bloc

CLB : Control Logic Bloc

VHDL : Very High Speed Integration Circuit Hardware Description Language

RNA : RiboNucleic Acid

DNA : DeoxyriboNucleic Acid

REVEAL : Reverse Engineering Algorithm

COD : Coefficient of Determination

DAG : Direct Acyclic Graph

RPP : Rapid Prototyping Platform

CMC : Canadian Microelectronics Corporation

ASIC : Application Specific Integrated Circuit

RAM : Random Access Memory

FIFO : First In First Out

PC : Personal Computer

AMBA : Advanced Microprocessor Bus Architecture

MSE : Mean Square Error

Pour l'amour de ma vie ma femme karima, pour son amour, son soutient et surtout pour sa patience

Pour mes très chers garçons Ramy et Samy

À la mémoire de mes parents.

Je dédie ce mémoire.

Remerciements

Je tiens, en premier lieu, à remercier le professeur El Mostapha Aboulhamid, mon directeur de recherches qui m'a accueilli dans son groupe de recherche et m'a accordé sa confiance. Je le remercie sincèrement de m'avoir soutenu et conseillé durant mon passage dans le laboratoire LASSO.

J'exprime également ma gratitude et reconnaissance à mon co-directeur de recherches, le professeur Jean Pierre David pour ses conseils, sa grande disponibilité tout au long du projet et surtout d'avoir supervisé ce travail.

Mes remerciements vont également à mes beaux parents Tabbi anneni Saad Zaghloul et Saadia pour leur soutient et leur affection.

Introduction

La modélisation des systèmes biologiques complexes est devenue aujourd'hui un domaine très convoité par de nombreux chercheurs, mathématiciens, biologistes et informaticiens. Ceux-ci tentent de comprendre le fonctionnement de ces systèmes et de découvrir les manières avec lesquelles leurs composants interagissent pour pouvoir les modéliser et ainsi les simuler. Plus précisément, la recherche génomique cherche à découvrir les principes qui régissent les composants génétiques. Elle vise ainsi à comprendre la façon par laquelle les cellules arrivent à contrôler et exécuter un très grand nombre d'opérations ainsi que pourquoi et comment les systèmes cellulaires échouent dans les maladies. Les mécanismes biologiques sont intrinsèquement parallèles et extraordinairement intégrés. Les chercheurs veulent comprendre la nature de la fonction cellulaire et comment les gènes et leurs produits forment collectivement un système biologique [1].

Il devient de plus en plus évident qu'il est nécessaire d'étudier le comportement des gènes d'une façon holistique plutôt qu'individuelle [2]. Les gènes ne sont pas isolés ou indépendants. Ils ne se comportent pas comme des êtres séparés et autonomes qui feraient leur travail chacun de leur côté indépendamment les uns des autres. Les réseaux de régulation génétique offrent un cadre conceptuel puissant permettant la modélisation, l'étude et la compréhension des interactions dans les systèmes biologiques. Cela exige inévitablement des méthodes mathématiques formelles et des outils informatiques puissants pour pouvoir traiter des quantités massives de données et pouvoir construire des modèles des interactions génétiques.

Il y a deux principaux secteurs de recherche:

- Le premier, vise à découvrir et à comprendre les mécanismes fondamentaux de la régulation des gènes en essayant d'inférer un modèle sur la base d'un certain nombre d'observations en laboratoire.
- Le deuxième, utilisant le modèle inféré, vise à faire des prédictions à l'aide d'analyse mathématique et de simulation informatique. Ces prédictions sont utilisées dans un but d'intervention, notamment pour traiter certaines maladies.

Les technologies récentes comme les puces d'ADN ou les biopuces permettent de mesurer simultanément le niveau d'expression de plusieurs milliers de gènes à un instant donné [24]. Plusieurs mesures peuvent être effectuées sur différentes cellules à différents moments et dans différentes conditions. Ces données d'expression de gènes ainsi extraites constituent une base de connaissance essentielle et nécessaire dans la compréhension des mécanismes de régulation génétique. Le processus de la modélisation des interactions génétiques commence avec la construction d'un modèle qui décrit le processus de régulation.

Il existe plusieurs approches de modélisation des réseaux génétiques :

- Le modèle graphique
- Le modèle bayesien
- Le modèle basé sur les équations différentielles
- Le modèle booléen

Le choix d'une classe de modèles devrait être fait selon les données d'entrées et le type de résultats recherchés. La fiabilité du résultat d'analyse génétique dépend fortement de la qualité et des caractéristiques de ces données.

Par exemple, un modèle avec beaucoup de paramètres peut être capable de capturer des phénomènes plus détaillés ou « de bas niveau » (comme des concentrations de protéines) mais il exigera de très grandes quantités de données.

Inversement, un modèle avec moins de paramètres et une complexité moindre réussira à capturer des phénomènes « de haut niveau » (comme par exemple si un gène est exprimé ou non-exprimé) et il exigera des quantités de données beaucoup plus petites [1]. Les modèles de régulation génétique peuvent varier en matière de détails biochimiques. Il existe différents niveaux de granularité avec lequel les différents modèles décrivent le processus de régulation (fin, moyen et gros) [17].

1.1 Modélisation des réseaux de régulation génétique

Il y a eu plusieurs tentatives pour modéliser les réseaux de régulation génétique. Une façon intuitive pour le faire est de voir le réseau de régulation génétique comme étant un graphe orienté [17] où les sommets représentent les gènes et les arcs les interactions entre ces gènes. Dans cette modélisation, la topologie de tout le réseau est contenue dans des bases de données. Pour des cas simples, la représentation graphique permet de répondre à beaucoup d'interrogations biologiques. Tout le système peut être représenté en mémoire et différents traitements peuvent être effectuées notamment la détection des cycles qui jouent un rôle très important dans la biologie [13]. Par contre pour des réseaux génétiques plus complexes (contenant au moins une centaine de gènes), le coût de recherche des composants du réseau dans les bases de données augmente d'une façon exponentielle comparativement à la taille du réseau. La représentation statique de tels systèmes devient pratiquement impossible en raison des contraintes mémoires.

La coopération des gènes représente une autre faiblesse pour cette modélisation. En effet, dès que deux ou plusieurs gènes coopèrent pour réguler l'expression d'autres gènes, il devient difficile de prévoir les valeurs de l'expression des gènes. Il est alors nécessaire de préciser la manière dont la co-régulation s'exerce sur les gènes. La présence des cycles imbriquées dans lesquels un gène peut prédire d'autres gènes ou aussi peut être prédit par sa propre expression et d'autre gènes, rend l'évaluation des propriétés dynamiques du réseau une tâche très difficile.

La représentation graphique est un modèle statique. En effet, les interactions génétiques sont connues et sont stockées dans des bases de données.

Les réseaux bayésiens constituent un autre modèle graphique qui a été utilisé dans la modélisation des interactions génétiques. C'est un modèle qui représente explicitement des rapports probabilistes entre les différents gènes [2, 17, 20].

Un réseau bayésien est un réseau causal auquel on ajoute des éléments de probabilités. Ce qui est important avec ce type de réseau, c'est le fait que les relations causales ne sont pas absolues mais sont associées à une probabilité indiquant le degré de confiance que l'on a dans l'évènement. L'incertitude est formalisée d'une part dans la probabilité d'un état d'un noeud qui dépend de l'information disponible et d'autre part dans les relations qui existent entre les noeuds par les probabilités conditionnelles. Les nœuds représentent les gènes et leurs niveaux d'expression. Le niveau d'expression de chaque gène est déterminé par une table de probabilité conditionnelle qui donne une probabilité au gène pour chacune des combinaisons des valeurs parentales. Les arcs représentent les interactions qui existent entres les gènes.

Les réseaux bayésiens permettent de traiter les aspects stochastiques d'expression de gène. De plus, ils peuvent être utilisés quand les informations sur le système sont incomplètes.

Bien que les réseaux bayesiens et les modèles graphiques soient des représentations intuitives des réseaux de régulation génétique, ils ont l'inconvénient de laisser de côté les aspects dynamiques de la régulation des interactions génétique. En effet, les réseaux bayésiens sont statiques [2]. Ils ne permettent pas les boucles alors qu'elles jouent un rôle très important dans les systèmes biologiques. Cependant, des généralisations dynamiques ont été proposées [19] pour représenter les boucles mais ces modèles exigent beaucoup de données [2, 19].

Dépendamment de la complexité du réseau, notamment le nombre de parents de chaque gène, les tables de probabilités conditionnelles associées à chaque gène peuvent contenir beaucoup de données et peuvent devenir très volumineuses. L'algorithme de recherche devient très coûteux.

L'inférence dans un système de cette nature devient une tâche très ardue. Comme le modèle précédent, les réseaux bayésiens rencontrent le problème de la dimension ou plus spécifiquement la taille du système. Il est pratiquement impossible de représenter statiquement de grands réseaux (une centaine de gènes).

Pour surmonter les insuffisances citées ci-dessus (en particulier la propriété statique) un autre formalisme dynamique a été utilisé pour permettre l'étude du comportement du réseau de

régulation de gènes à long terme. Parmi les méthodes de modélisation dynamique, la plus utilisée dans la biologie est sans doute le modèle basé sur les équations différentielles.

Les modèles basés sur les équations différentielles représentent l'approche qui adopte une vue plus détaillée sur les réseaux de régulation génétiques. Dans ce modèle, on représente l'évolution continue du système. Pour cela, on modélise le réseau par un système différentiel où les variables sont les concentrations de protéines. Dans la plupart des situations biologiques, les interactions considérées ne sont pas linéaires. Ceci conduit à des modèles différentiels généralement impossible à résoudre d'une façon analytique. Souvent des approches numériques sont utilisées.

Cette modélisation a bien sûr ses limites. L'inconvénient majeur reste la taille du système qui peut devenir très grande. Un autre inconvénient réside dans le fait que l'évolution de tels réseaux en fonction du temps peut devenir problématique en raison des longs intervalles de temps qui peuvent affecter l'expression des gènes [17, 18].

Un modèle qui a reçu beaucoup d'intérêt par les chercheurs est le modèle booléen. En effet, les systèmes discrets semblent plutôt bien adaptés aux systèmes biologiques car ces derniers présentent des phénomènes facilement modélisables par des variables discrètes. En d'autres termes, l'abstraction booléenne simplifie énormément la tâche de modélisation. On veut étudier le comportement régulateur collectif sans détails quantitatifs spécifiques [1, 2].

Dans le modèle booléen, un gène est exprimé ou non-exprimé. Il est donc représenté par une variable booléenne qui vaut 1 ou 0. Les différentes interactions qui existent entre les gènes sont modélisées par des fonctions booléennes. L'avantage d'une telle représentation est bien sûr sa simplicité. Pour représenter un réseau booléen, on peut utiliser un graphe orienté pour voir la topologie du réseau (influence des nœuds les uns sur les autres) ainsi que des tables de vérité pour expliciter les fonctions booléennes.

Dans ces réseaux, chaque gène (cible) est prédit par plusieurs autres gènes au moyen d'une fonction booléenne (prédicteur). Il est facile de voir que si on connaît les valeurs des gènes prédictifs, on peut déduire la valeur du gène cible. Conceptuellement ce déterminisme semble problématique [1, 2]. La supposition d'une seule règle logique/gène peut nous induire à des conclusions erronées sachant que ces règles elle mêmes sont inférées à partir de données

expérimentales. Pour surmonter la rigidité des réseaux booléens, des chercheurs ont introduit une nouvelle classe de modèles appelée les réseaux probabilistes booléens (PBNs) [1].

Les PBNs sont des généralisations probabilistes des réseaux booléens standards. Ils offrent un cadre de modélisation flexible et puissant [1, 2 ,3].

Les PBNs partagent les propriétés attirantes des réseaux booléens. Ils permettent l'étude systématique de la dynamique du réseau [2]. Cependant, à cause de leur nature probabiliste, ils sont capables de prendre en compte le non-déterminisme qui est intrinsèque aux systèmes biologiques [1, 2].

Les PBNs représentent un compromis entre le déterminisme absolu des réseaux booléens et la nature probabiliste des réseaux bayésiens. L'idée fondamentale est de combiner plusieurs prédicteurs ou fonctions booléennes de sorte que chacun puisse apporter une contribution à la prévision d'un gène cible.

Le PBN offre un cadre conceptuel puissant pour modéliser les interactions génétiques réelles. En effet, le PBN permet de modéliser avec facilité les boucles. La puissance de la logique booléenne permet au PBN de résoudre le problème de co-régulation vu dans le modèle graphique.

Les réseaux génétiques délivrent et prédisent des quantités énormes d'informations concernant le système génétique. Un des plus importants problèmes est de découvrir les points stables (attracteurs) d'un système spécifique. Les attracteurs peuvent être vus comme un sous-ensemble d'états souvent revisités, c'est-à-dire des cycles dans un graphe.

Ils ont une longueur définie comme étant le nombre d'états de ce cycle. En génétique les attracteurs ont beaucoup d'interprétations. Ils peuvent exprimer la stabilité du système face aux perturbations externes. Ils correspondent à une spécialisation cellulaire ou un cycle méiotique de cellule ou aussi à l'état d'une cellule [13]. Il a été noté que les attracteurs sont très courts dans le monde réel comparativement à une croissance exponentielle dans un régime chaotique [13]. Tout cela montre l'importance de découvrir les attracteurs dans un réseau de régulation de gènes. L'exécution de cette tâche demande d'importantes ressources logicielles et matérielles.

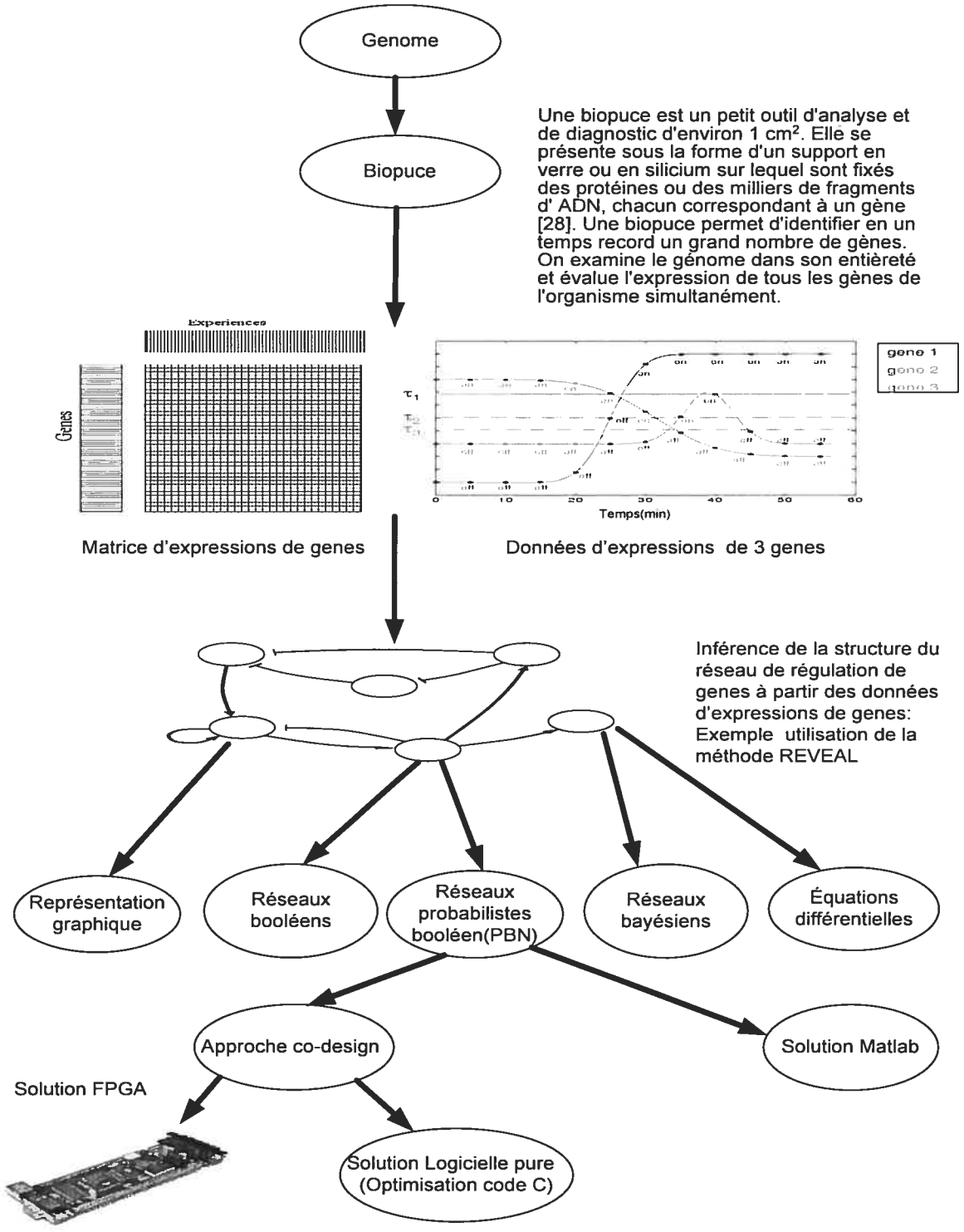


Figure 1 : flot des tâches

La détection des attracteurs suit le cheminement de la figure 1. D'abord les données d'expressions de tous les gènes de l'organisme étudié doivent être déterminées et stockées en utilisant la technique « biopuce ». L'étape suivante consiste à inférer la structure du réseau de régulation génétique à l'aide d'une méthode d'inférence de la structure comme par exemple REVEAL ou la méthode basée sur l'utilisation des coefficients de détermination. Dépendamment de la qualité des données d'inférence et des résultats recherchés, un modèle est choisi pour modéliser les interactions génétiques. Finalement, notre système peut être utilisé pour détecter les attracteurs.

1.2 Problématique

La taille du système représente un des principaux obstacles dans la modélisation de la régulation génétique. En effet, le graphe de transition d'états qui représente le système a un nombre exponentiel de noeuds comparativement au nombre de gènes considérés dans le système. La construction statique d'une table représentant un tel graphe peut devenir pratiquement impossible (problème de mémoire). En effet, dans l'approche classique où la solution est purement logicielle [16], on ne peut manipuler plus de 15 gènes en raison des contraintes mémoire. Pour 15 gènes, l'ordre de grandeur de la taille mémoire atteint les milliards d'octets.

1.3 Objectif

Notre objectif est de concevoir un système capable de modéliser un réseau de régulation génétique de taille réaliste (au moins quelques centaines de gènes). Ce système doit être capable de détecter les attracteurs en utilisant le PBN comme un modèle de régulation des interactions génétiques.

1.4 L'approche

Afin d'atteindre notre objectif en évitant le problème évoqué plus haut, nous présentons dans ce mémoire une solution très efficace qui utilise l'approche de codesign matériel/logiciel. Cette approche est basée sur l'utilisation des FPGAs.

La solution proposée ne construit pas d'une façon explicite la table représentant le graphe de transition mais elle utilise les fonctions booléennes comme une représentation implicite. Avec un FPGA de taille moyenne (Virtex v2000efg680-6, Xilinx) s'exécutant à 25 Mhz, on a implémenté un réseau de 1000 gènes avec 2 prédicteurs par gène et 5 variables par prédicteur. La puissance de calcul a atteint l'ordre de 50 milliards d'équations logiques par seconde, ce qui montre la puissance et l'efficacité de la solution proposée.

1.5 Plan

Dans le chapitre 2, on présente comment les différents modèles de régulation modélisent les interactions génétiques avec leurs avantages et inconvénients. On présente deux méthodes d'inférence de la structure du réseau à partir des données d'expression de gènes. L'une est basée sur les coefficients de détermination et l'autre utilise l'entropie de Shannon. À la fin de ce chapitre, on donnera une introduction au FPGA, son architecture interne, comment un FPGA est programmé ainsi que les avantages de ce type de circuits. On présente aussi quelles sont les raisons qui nous ont poussé à utiliser ce genre de circuit pour résoudre le problème évoqué ci-dessus.

Le chapitre 3 présente en détail le PBN (Probabilistic Boolean Network), le modèle de régulation génétique sur lequel notre système matériel-logiciel est implémenté.

Le chapitre 4 est consacré entièrement à la description de la partie implémentation. On décrit les différentes étapes qui nous ont permis de réaliser ce système. On présente la plateforme qui a supporté notre approche de co-design. On montre la partie matérielle qui est implémentée dans le FPGA et la partie logicielle. La partie logicielle est composée de deux parties, celle qui permet de générer le code de la partie matérielle dans une phase d'initialisation et celle qui interagit avec la partie matérielle pendant la simulation du réseau.

Dans le chapitre 5, on présente les résultats des expérimentations effectuées et les performances de notre système.

Enfin, une conclusion de notre approche sera donnée dans le chapitre 6. Ce travail pourrait aller plus loin. Quelques perspectives sont également proposées.

L'état de L'art

Pour comprendre le fonctionnement d'un organisme biologique quelconque, il est nécessaire d'étudier les différentes interactions qui existent entre ses différents composants. Ce sont ces interactions qui régissent l'activité de la cellule afin de lui permettre de s'adapter en permanence aux variations de son environnement. Il existe de véritables cascades d'interactions faisant souvent intervenir des boucles de rétroaction positives et négatives. Les réseaux de régulation génétiques permettent de décrire les interactions qui existent entre les différents gènes et protéines d'un organisme. Plus spécifiquement, ils permettent de montrer quels sont les gènes qui sont responsables dans la régulation d'autres gènes et comment l'environnement externe peut affecter le niveau d'expression d'un gène donné.

Pour pouvoir étudier ces interactions, il est impératif de commencer par la modélisation formelle du réseau de régulation. Différents modèles formels ont été utilisés pour la modélisation des interactions génétiques. Ils se divisent principalement en deux catégories: les systèmes discrets et les systèmes continus. Parmi ces modèles, il y a la représentation graphique, les réseaux bayésiens, les modèles basés sur les équations différentielles et le modèle booléen.

En général, il est difficile de dire lequel des modèles est meilleur. Cependant, on peut faire des comparaisons [17] en prenant en considération quelques aspects importants tels que les modèles sont discrets ou continus, statiques ou dynamiques, déterministes ou stochastiques et qualitatifs ou quantitatifs. Aussi, des comparaisons peuvent être faites au niveau de la granularité avec laquelle ils décrivent le processus de régulation. Ce niveau de granularité permet de déterminer le coût de traitement en considérant la quantité de données qui doit être introduite pour pouvoir faire des simulations [17].

2.1 La représentation graphique

Soit un système formé de 3 gènes (x_1, x_2, x_3), les interactions qui existent dans ce système sont définies selon le graphe donné dans la figure 2.

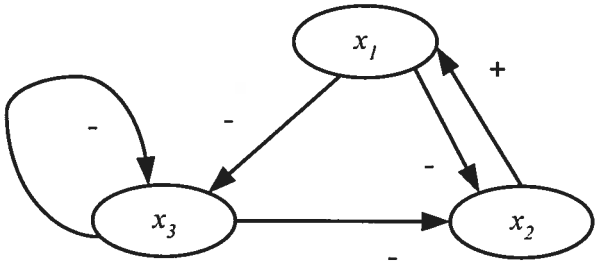


Figure 2 : Graphe orienté

Pour modéliser ce système, la façon la plus intuitive est de le considérer comme une représentation graphique dans laquelle les sommets représentent les gènes et les arcs correspondent aux différentes interactions entre les gènes. D'un point de vue mathématique une telle représentation peut être considérée comme un graphe orienté.

2.1.1 Définition et notation

Soit G un graphe orienté défini comme un ensemble (N, A) où N est l'ensemble des nœuds et A l'ensemble des arcs. On définit un arc orienté comme un couple (t, q) de sommets où t représente la tête et q correspond à la queue de l'arc.

Dans le graphe de la figure 2, le nœud représente le gène et les arcs correspondent aux différentes interactions qui existent entre les différents gènes [17].

Les nœuds et les arcs peuvent être étiquetés afin de donner un sens aux différentes interactions qui existent entre les différents gènes. On définit alors un arc orienté comme étant (t, q, s) où s peut être égal à « + » ou « - ».

Cela veut dire que t peut être activé (+) ou inhibé (-) par q. Dans la figure 2, les ensembles N et A sont définis comme suit :

$$N = \{x_1, x_2, x_3\}$$

$$A = \{(x_2, x_1, -), (x_1, x_2, +), (x_2, x_3, -), (x_3, x_3, -)\}$$

2.1.2 Régulation des interactions génétiques

En étudiant ce type de modèle, la première propriété qu'on peut tirer est sa simplicité dans la modélisation. En effet, avec cette représentation graphique on peut modéliser facilement des systèmes simples et alors plusieurs interrogations biologiques peuvent trouver leurs réponses. La topologie de tout le réseau est contenue dans une base de données. Il existe plusieurs bases de données. On peut citer par exemple, KEGG [29] et GeneNet [30]. Des applications spécifiques peuvent être utilisées pour permettre à l'utilisateur de traiter ou visualiser une partie ou tout le réseau [17]. La connectivité (nombre de régulateurs par gène) peut nous indiquer la complexité du réseau. Généralement pour des réseaux complexes, les tailles des différentes bases de données sont gigantesques et le coût de traitement est élevé. En effet, le coût en matière de recherche pour la représentation des composants du réseau et celui de la détection des cycles dans le réseau deviennent exponentiels comparativement à la taille du réseau contenu dans la base de données. La représentation statique de systèmes plus complexes (qui représentent les réseaux génétiques réels) devient pratiquement impossible à cause des contraintes mémoires. La coopération des gènes dans la prédiction d'autres gènes pose un sérieux problème pour cette représentation. Regardons maintenant comment se comporte ce modèle lorsque deux ou plusieurs gènes coopèrent pour réguler l'expression d'un autre gène. Voyons par exemple quelle serait l'expression du gène x_3 dans la figure 3 qui dépend de sa propre expression et celle du x_1. Aussi la valeur du gène x_2 dépend de l'expression de x_1 et celle du gène x_3. L'absence de x_1 et/ou la présence de l'expression de x_3 sont-elles suffisantes pour que le gène x_3 (resp. x_2) soit exprimé ? Il devient alors nécessaire de préciser la manière dont la co-régulation s'exerce sur le gène en question.

Dès que les cascades de co-régulation deviennent importantes, il devient très difficile de prédire le comportement dynamique du système. C'est le cas notamment des cascades de régulation bouclées sur elles-mêmes où l'expression de chaque gène est directement influencée par d'autres gènes et influençant directement d'autres gènes, dépend aussi indirectement (ou directement) de sa propre expression préalable via les autres gènes de la boucle. Dès que

plusieurs circuits [1] sont imbriqués, il devient alors très difficile d'évaluer les propriétés dynamiques du réseau.

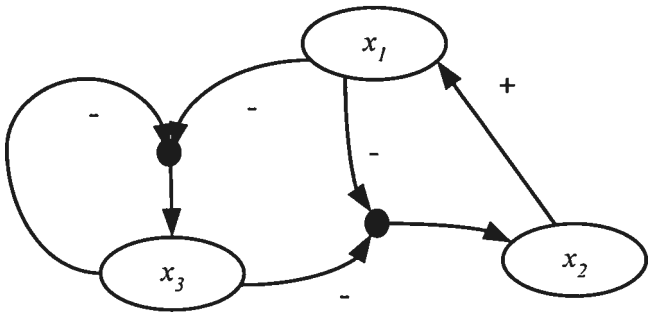


Figure 3 : Hypergraphe orienté

Le recours aux hypergraphes peut résoudre le problème de la co-régulation. En effet, dans un hypergraphe orienté, un hyper arc est défini comme un triplet (q, R, S) où R représente l'ensemble de gènes régulateurs et S une liste correspondante de signes qui indiquent l'influence de ces gènes [17].

Dans la figure 3, l'ensemble A est défini comme suit :

$$A = \{(x_2, [x_1, x_3], [-, -]), (x_1, [x_2], [+]), (x_3, [x_1, x_3], [-,-])\}$$

Cette façon de faire, nous permet d'écrire d'une manière explicite les différentes relations qui entrent dans la co-régulation. Il faut noter aussi que même dans le cas des hypergraphes orientés, la topologie du réseau est contenue dans une base de données.

La représentation graphique offre une manière statique et déterministe pour modéliser les interactions génétiques. Les interactions entre les différents gènes du système sont connues et sont statiques. Pour des réseaux simples, cette modélisation offre un cadre de modélisation simple et interprétable. Cependant avec des réseaux plus complexes, les tâches de modélisation et de détection des boucles exigent beaucoup de ressources et des coûts de traitements très élevés.

De plus, l'état de l'expression d'un gène cible à l'instant t+1 est déterminé exclusivement par l'état de l'expression d'un ou plusieurs gènes (prédicteurs) à l'instant t.

[1] On entend ici par circuit, un ensemble de transitions d'états cyclique

Ce déterminisme rend la modélisation trop rigide car les prédicteurs sont établis par expérimentation et la confiance qu'on peut avoir dans leur exactitude n'est jamais totale.

2.2 Réseaux bayésiens

2.2.1 Définition et notation

Soit G un graphe acyclique orienté (figure 4) formé de cinq variables aléatoires x_1, x_2, x_3, x_4, x_5. On dit que x_1 et x_2 sont indépendants si $P(x_1, x_2) = P(x_1)P(x_2)$ pour tout x_1 *et* x_2, $P(x_1, x_2)$ étant la distribution jointe de x_1 et x_2. Sinon elles sont dites dépendantes. Cette dépendance peut être représentée dans le graphe G par un arc entre les deux variables aléatoires. Dans ce cas, on parle de relation parentale. Dans le cas de la figure 4, on dit que x_1 est le parent de x_3. Si n variables aléatoires $x_1, x_2, ..., x_n$ sont indépendantes, la probabilité jointe peut être déterminée par $P(x_1, x_2, ...x_n) = \prod_{i=1}^{n} P(x_i)$. Elle permet de spécifier les probabilités de toutes les combinaisons de valeurs des variables aléatoires. Dans ce cas la probabilité de distribution jointe est dite complète, elle est formée par 2^n probabilités.

On définit la relation entre x_1 et x_5 comme une relation d'indépendance conditionnelle qu'on note $P(x_1 | x_3, x_5) = P(x_1 | x_5)$. Cette indépendance conditionnelle peut être dénotée comme $I(x_1; x_5 | x_3)$ où x_1 est indépendante de x_5 sous la condition x_3.

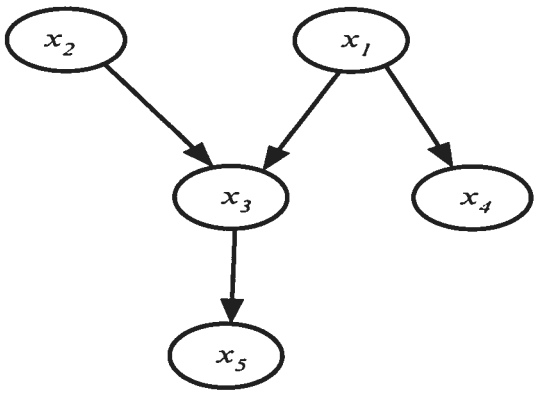


Figure 4: Graphe orienté

Un réseau bayésien (ou probabiliste) est défini par un graphe G orienté dont les nœuds représentent des variables aléatoires et les arcs correspondent aux différentes relations entre ces noeuds. Chaque nœud dépend directement de ses parents et est indépendant de ses non-descendants. Soit $p_0,...,p_k$ les parents du nœud x_i qu'on note $Parents(x_i)$.

On associe à chaque nœud x_i une table de probabilité conditionnelle (TPC) qui quantifie les effets de ses parents. C'est une table de 2^k entrées, chacune représente une configuration des parents. De point de vue qualitatif, la structure des réseaux bayésiens décrit les rapports entre ces variables sous forme de relations conditionnelles indépendantes. Au niveau quantitatif, les rapports entre les différentes variables peuvent être décrites par des probabilités de distribution conditionnelles.

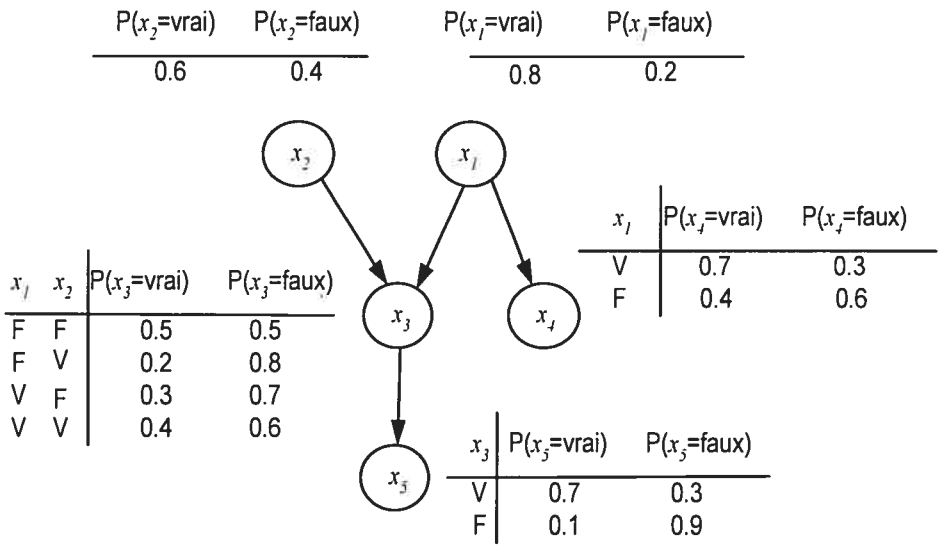


Figure 5 : Un réseau bayésien.

La structure des réseaux bayésiens nous permet de répondre à plusieurs interrogations et ainsi déterminer leurs probabilités. Supposons qu'on veut savoir quelle est la probabilité du gène x_3 d'être présent quelles que soient les valeurs des gènes x_1 et x_2 dont il est dépendant. Pour cela, on doit utiliser le théorème des probabilités totales. Soit un système complet d'évènements $x_1,x_2,...,x_k$, pour un évènement quelconque x on a : $P(x)=\sum_{k=1}^{n}P(x\cap x_k)$.

Or pour tout k : $P(x\cap x_k)=P(x|x_k).P(x_k)$, on a donc $P(x)=\sum_{k=1}^{n}P(x|x_k).P(x_k)$.

En appliquant donc le théorème des probabilités totales et en utilisant les tables de probabilités conditionnelles de la figure 5, $P(x_3)$ peut être calculée de la manière suivante :

$$P(x_3) = \sum_{i=1}^{n=4} P(x_3 | Parents(x_3)).P(Parents(x_3))$$ (n : représente le nombre d'évènements engendrés par x_1 et x_2).

$$P(x_3) = (P(x_3|x_1, x_2).P(x_1, x_2)) + (P(x_3|x_1, \neg x_2).P(x_1, \neg x_2)) + (P(x_3|\neg x_1, x_2).P(\neg x_1, x_2)) + (P(x_3|\neg x_1, \neg x_2).P(\neg x_1, \neg x_2))$$

$$P(x_3) = (0.64 * 0.4) + (0.12 * 0.3) + (0.32 * 0.2) + (0.08 * 0.5) = 0.256 + 0.036 + 0.064 + 0.040 = 0.394$$

Donc, $P(\neg x_3) = 1 - 0.394 = 0.606$

Aussi, la probabilité jointe de tous les nœuds du graphe (figure 5) peut être définie comme suit :

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_5|x_3, x_4, x_2, x_1).P(x_4|x_1, x_2, x_3).P(x_3|x_2, x_1).P(x_2, x_1)$$

où $P(x_2, x_1) = P(x_2).P(x_1)$

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_5|x_3, x_4, x_2, x_1).P(x_4|x_1, x_2, x_3).P(x_3|x_2, x_1).P(x_2).P(x_1)$$

En utilisant les relations d'indépendances conditionnelles, on peut simplifier l'équation de la façon suivante :

On supprime x_4, x_1 et x_2 de $P(x_5|x_3, x_4, x_2, x_1)$ puisque x_5 est indépendante de ces trois variables étant donnés son parent (ici x_3).

On supprime x_2 et x_3 de $P(x_4|x_1, x_2, x_3)$ puisque x_4 est indépendante de deux variables étant donnés son parent (ici x_1).

Donc, la probabilité de distribution jointe peut être écrite sous la forme simplifiée suivante :

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_5|x_3).P(x_4|x_1).P(x_3|x_1, x_2).P(x_2).P(x_1)$$

La structure des réseaux bayésien reflète les relations d'indépendances et les indépendances conditionnelles. Ces relations nous permettent de représenter la distribution jointe d'une façon plus compacte en réduisant le nombre de paramètres.

La distribution jointe complète de probabilité est définie comme étant le produit des distribution conditionnelles locales :

$$P(x_1, x_2, ...x_n) = \prod_{i=1}^{n} P(x_i | Parents(x_i))$$

Si nous avons n nœuds binaires, la distribution jointe complète exigerait $O(2^n)$ probabilités, par contre la forme factorisée ne demanderait que $O(n*2^k)$ probabilités où k est le nombre maximum d'entrées par nœud. Prenons l'exemple de la figure 5, la table de distribution jointe complète correspondante à un réseau composé de 5 variables aléatoires exigerait $2^5-1=31$ nombres. Alors qu'avec une structure imposée par les relations d'indépendances conditionnelles, la table de distribution jointe est réduite à 10 nombres. L'indépendance conditionnelle permet de diminuer la grandeur des tables de distribution jointe. Dans l'exemple de la figure 5, on a remplacé une grande table par 5 petites tables plus faciles à calculer.

Par exemple supposons qu'on veut déterminer la probabilité de la proposition suivante :
« Les gènes x_1 et x_3 sont présents, tous les autres gènes sont absents ».

En prenant les probabilités conditionnelles locales définies dans les TPC de la figure 5, la probabilité correspondante à la proposition est égale à :

$$P(x_1 = v, x_2 = f, x_3 = v, x_4 = f, x_5 = f) = P(x_1).P(x_3|x_1, \neg x_2).P(\neg x_5|x_3).P(\neg x_4|x_1).P(\neg x_2)$$
$$P(x_1 = v, x_2 = f, x_3 = v, x_4 = f, x_5 = f) = 0.8 * 0.3 * 0.3 * 0.3 * 0.4$$
$$P(x_1 = v, x_2 = f, x_3 = v, x_4 = f, x_5 = f) = 0.00864$$

2.2.2 Application aux interactions génétiques

En appliquant ce modèle à l'inférence des réseaux de régulation de gènes, on associe les nœuds avec les gènes et leurs niveaux d'expressions alors que les arcs représentent les interactions qui existent entre les différents gènes. Par exemple, la structure du réseau de la figure 4 fait que le gène x_1 et x_2 régulent le gène x_3 qui à son tour régule le gène x_5. Aussi, le gène x_1 régule x_4. L'expression de chacun de ces gènes peut être déterminée en considérant les différentes tables de probabilités conditionnelles associées à chaque gène.

Le réseau bayésien offre une façon simple et unique pour étendre la probabilité de distribution en termes de probabilités conditionnelles simples. L'avantage de cette composition est qu'un système complexe d'interactions peut être vu comme une composition de plusieurs sous systèmes facilitant l'interprétation et la compréhension de ces interactions [20].

L'avantage d'utiliser les réseaux bayésiens pour la modélisation de la régulation génétique est qu'ils sont facilement interprétables, ils spécifient clairement les dépendances entre les gènes.

L'inconvénient majeur des réseaux bayésiens est qu'ils sont statiques [1, 17], la propriété acyclique du graphe ne permet pas d'avoir de cycles directs, or ces derniers ont une signification très importante dans la biologie (différentiation, prolifération, etc.). Cependant, le recours aux réseaux bayésiens dynamiques [19] peut résoudre le problème des boucles.

Un des plus importants problèmes dans la modélisation de la régulation génétique est connu sous le nom « problème de dimension ». En effet, la « dimension » ou taille du système qui peut devenir exponentielle, est un obstacle pour la modélisation bayésienne. La représentation statique de tels systèmes devient pratiquement impossible à réaliser en considérant les ressources statiques dont on dispose.

On peut prendre par exemple un réseau composé d'une centaine de gènes dans lequel le nombre moyen des parents par gène est grand (une dizaine par exemple). La taille des différentes TPC devient très grande. La modélisation statique de ce réseau devient difficile.

2.3 Le modèle basé sur les équations différentielles

2.3.1 Définition et notation

Parmi les méthodes de modélisation dynamique continue, la description différentielle est la plus utilisée en biologie. Les modèles des équations différentielles représentent l'approche qui adopte une vue plus détaillée sur les réseaux régulateurs génétiques. Le formalisme basé sur les équations différentielles modélise les concentrations des ARNs, protéines et autres molécules à l'instant t par des variables continues $x_i(t)$ ayant des grandeurs réelles positives [17, 18]. Si un réseau est composé de n gènes, alors la modélisation des équations différentielles devrait être composée de n équations différentielles, une pour chaque gène. La forme générale d'une équation différentielle pour le i^{eme} gène peut être :

$$\frac{dx_i}{dt} = f_i(x_1, x_2, ...x_n), \quad 1 \le i \le n$$

Dans la plupart des situations biologiques, les interactions considérées sont non linéaires.

Les fonctions f_i sont donc généralement non linéaires. Elles définissent les variations des concentrations des protéines comme étant la différence entre le terme de la synthèse et celui de la dégradation de la protéine.

La fonction f_i peut être définie comme:

$$f_i(x_1, x_2, ...x_n) = S(x_1, x_2, ...x_n) - D(x_1, x_2, ...x_n) \quad 1 \le i \le n$$

où $S(x_1, x_2, ...x_n)$ représente la synthèse et $D(x_1, x_2, ...x_n)$ représente la dégradation de protéine.

2.3.2 Modélisation de la régulation génétique

La figure 6 montre un réseau de régulation génétique formé de 3 gènes (x_1, x_2, x_3). Chaque gène code une protéine qui contrôle l'expression d'un gène cible. On peut voir par exemple que les gènes x_1 et x_3 coopèrent et codent des protéines qui inhibent l'expression du gène x_2 et le gène x_2 code une protéine qui active l'expression du gène x_1. L'activation et l'inhibition de l'expression d'un gène est indiquée par « + » et « - ».

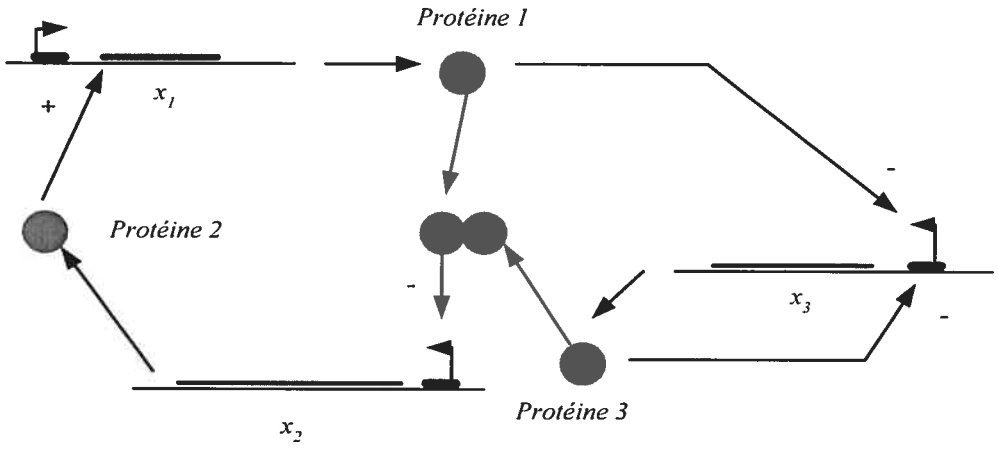


Figure 6: Réseau de régulation génétique [26]

L'état des variables (gènes) dans le modèle correspond à la concentration des protéines codées par les gènes dans le réseau.

Les interactions de régulation sont modélisées par des « fonctions d'étapes ». Celles-ci sont définies de la manière suivante [17] :

$$s^+(c_j, n_j) = \begin{cases} 1, & c_j > n_j \\ 0, & c_j < n_j \end{cases}$$

où c_j est la concentration de protéine et n_j est un seuil.

Une liste de seuils est définie pour chaque variable. Le seuil indique la concentration avec laquelle une protéine régulatrice peut influencer l'expression d'un gène cible ou non. Comme une protéine peut être impliquée dans plusieurs interactions régulatrices, elle peut avoir plusieurs seuils. Dans le cas de la protéine 3, il y a deux seuils n_{23} (le gène 2 est le gène cible) et n_{33} (le gène 3 est le gène cible) résultant de deux interactions dans lesquelles la protéine codée par le gène 3 est impliquée. Si deux seuils sont égaux, alors on leur donne le même nom n_3 [26]. Le terme synthèse est défini comme la somme des expressions des fonctions d'étapes [17, 26], chacun est pondéré par un paramètre de production. Il spécifie la logique de régulation du gène. Par exemple dans la figure 6, si la concentration c_2 de *la protéine* 2 atteint une valeur au dessus du niveau de seuil n_{12}, alors le gène x_1 sera exprimé avec un taux de production p_{12} sachant que la définition du terme de synthèse est $p_{12}*s^+(c_2, n_{12})$ où $s^+(c_2, n_{12})$ est la fonction d'étape [26]. Aussi, si la protéine 1 et la protéine 3 atteignent des valeurs au dessus de leurs niveaux de seuil respectifs n_{21} et n_{23} alors l'expression de la fonction d'étape sera évalué à zéro et l'expression du gène x_2 sera inhibé, sinon elle sera exprimée avec un taux de production p_{213} sachant que le terme synthèse est égal à $p_{213}*(1 - s^+(c_1, n_{21})* s^+(c_3, n_{23}))$ où $s^+(c_1, n_{21})$ et $s^+(c_3, n_{23})$ sont les fonctions d'étapes.

Le terme dégradation est déterminé soit par la multiplication du paramètre de dégradation et celui de la concentration g_3*c_3, ou par la multiplication d'une autre fonction d'étape et le coefficient de la concentration $g_{3+}\ g_{32}*\ s^+(c_2, nd_{32}))*c_3)$. Dans ce cas, la dégradation de la protéine est régulée par d'autres protéines [26].

Dans la figure 7 la coopération de la protéine 2 avec la protéine 3 favorise la dégradation de la protéine 3. C'est-à-dire tant que c_2 reste au-dessous de son niveau de seuil nd_{32}, la protéine 3 est dégradée avec un taux g_3. Tandis que si c_2 est au-dessus de ce seuil, elle est dégradée avec un taux g_3+g_{32}.

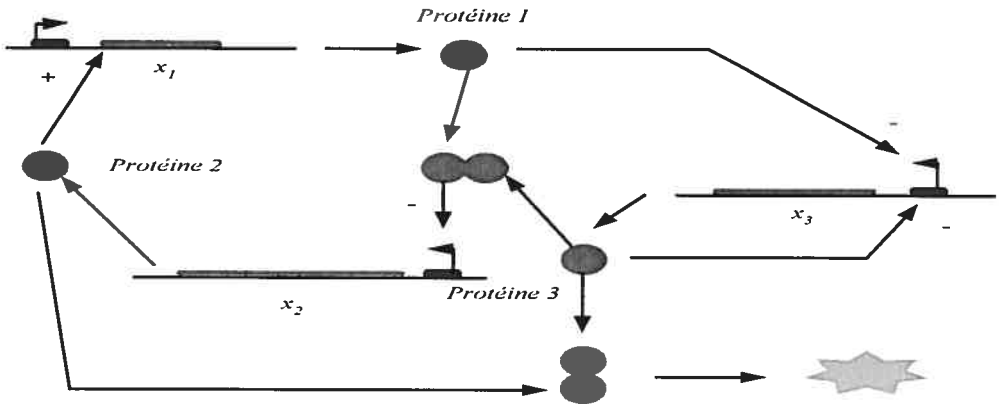


Figure 7 : Dégradation de la protéine 3 [26]

L'inconvénient majeur de ce modèle est le fait que la dimension (taille) du système peut croître d'une façon considérable et très vite. Généralement, il est impossible de résoudre de façon analytique ces systèmes différentiels non linéaires. On a alors recours aux simulations numériques.

2.4 Formalisme booléen

Le modèle qui a reçu beaucoup d'intérêt par les chercheurs est le modèle booléen et cela pour différentes raisons :

- L'abstraction booléenne simplifie énormément la tâche de modélisation.
- Les chercheurs veulent étudier le comportement régulateur collectif sans détails quantitatifs spécifiques.

Le formalisme des réseaux booléens s'intéresse aux principes fondamentaux génériques plutôt qu'aux détails biochimiques quantitatifs. Il permet de capturer les comportements dynamiques du réseau de régulation génétique. Il fournit le potentiel pour découvrir de nouvelles cibles pour des médicaments de certaines maladies comme par exemple le cancer [4]. Le modèle booléen a été introduit par Kauffman depuis plus de 30 ans. Dans ce modèle l'expression du gène est quantifiée uniquement par deux niveaux : « 1» (exprimé) et « 0» (non exprimé).
Le niveau d'expression (état) de chaque gène est relié aux états des autres gènes au moyen de règles logiques [1].

La figure 8, montre un diagramme logique décrivant l'activité génétique d'un exemple de réseau formé de 3 gènes. Il est caractérisé par les équations logiques suivantes :

$$f_1 = x_1 + x_2,\ f_2 = x_1 + x_2,\ f_3 = \overline{x_1 x_2} + x_1 x_2 \overline{x_3}$$

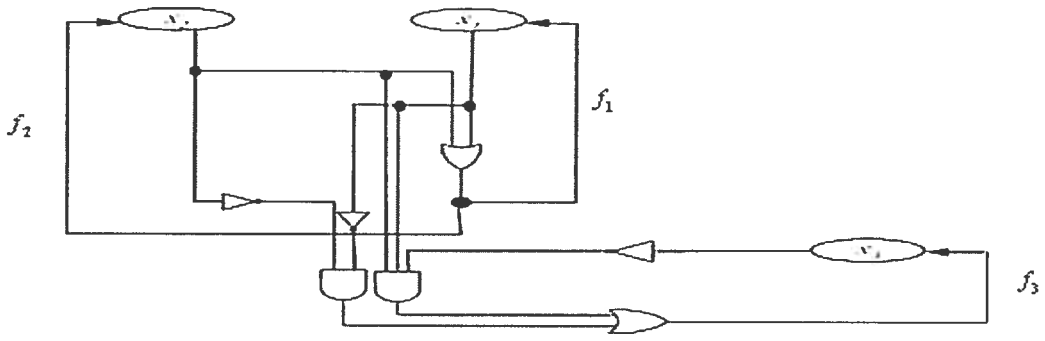


Figure 8 : Diagramme logique

Dans les sections qui suivent, nous donnons les définitions de base nécessaires pour la compréhension des notions qui seront introduites ainsi que toutes les notations des réseaux booléens et des réseaux booléens probabilistes. Pour plus de détails le lecteur pourra se référer à [1, 2, 3, 13].

2.5 Réseaux booléens

2.5.1 Définition : Un réseau booléen G(V, F) est défini par un ensemble de nœuds(gènes) $V = \{x_1, x_2, ..., x_n\}$ et une liste de fonctions booléennes $F = \{f_1, f_2, ..., f_n\}$ qui définit une topologie d'arrêtes. n est appelée taille ou dimension du réseau.

Chaque $x_i \in \{0, 1\}$ i = 1,..., n est une variable binaire et sa valeur à l'instant $(t+1)$ est complètement déterminée par la valeur des autres gènes $x_{j_{1(i)}}, x_{j_{2(i)}}, ..., x_{jk_i(i)}$ à l'instant t par la fonction booléenne $f_i \in F$. Chaque gène x_i est influencé par k_i gènes. Ils permettent de déterminer son état à l'aide de la fonction booléenne qui lui est associée. Donc pour k = 1,...,k_i on peut écrire que :

$$x_i(t+1) = f_i(x_{j1(i)}(t), x_{j2(i)}(t), ..., x_{jk_i(i)}(t)) \qquad (1)$$

Chaque nœud $\in \{0, 1\}$ représente l'état (expression) du gène i

$$x_i = \begin{cases} 1 & \text{signifie que le gène } i \text{ est exprimé} \\ 0 & \text{signifie que le gène } i \text{ n'est pas exprimé} \end{cases}$$

La liste de fonctions booléennes F représente les règles possibles d'interactions de régulation entre les gènes. Chaque fonction f détermine la valeur d'un nœud selon la valeur des autres nœuds qui l'influencent. $K = max_i\, k_i$ représente la connectivité maximum du réseau booléen. Les fonctions booléennes f_i peuvent être définies par des tables de vérité.

Pour représenter un réseau booléen, on peut utiliser un graphe orienté pour voir la topologie du réseau (influence des nœuds les uns sur les autres) ainsi que des tables de vérité pour expliciter les règles. Aussi, on peut représenter un réseau booléen par un graphe d'états. L'état du système est donné par les valeurs x_i pour chaque gène i à un instant donné. Grâce à cette représentation, nous voyons la trajectoire que le réseau emprunte.

Dans un réseau booléen, le nombre d'états est fini. Il existe dans le réseau au moins un attracteur qui peut être soit un cycle (états finaux périodiques), soit un point (états finaux statiques). La mise à jour de l'ensemble des gènes du réseau se fait d'une façon synchrone par l'intermédiaire des fonctions qui leur sont assignées. On peut voir facilement que la dynamique du réseau peut être déterminée complètement par (1).

Prenons un exemple : la figure 9 montre un réseau booléen composé de 3 gènes (x_1, x_2, x_3) avec les fonctions booléennes correspondantes données dans les tables de vérité (voir table I), la connectivité maximale du réseau est $K = 3$.

$$f_1 = x_1 + x_2 \qquad \text{// prochain } x_1$$

$$f_2 = x_1 + x_2 \qquad \text{// prochain } x_2$$

$$f_3 = \overline{x_1 x_2} + x_1 x_2 \overline{x_3} \qquad \text{// prochain } x_3$$

Table *I* : Tables de vérité pour les fonctions booléennes avec 3 gènes

$x_1\ x_2\ x_3$	f_1	f_2	f_3
0 0 0	0	0	1
0 0 1	0	0	1
0 1 0	1	1	0
0 1 1	1	1	0
1 0 0	1	1	0
1 0 1	1	1	0
1 1 0	1	1	1
1 1 1	1	1	0

La dynamique de ce réseau booléen est montrée dans la figure 9. Puisque notre réseau est composé de 3 gènes, alors notre système est formé de $2^3 = 8$ états possibles dans lesquels le réseau peut être à tout instant. Dans la figure 9, chaque état est représenté par un cercle et les flèches entre les états montrent les transitions du réseau selon les fonctions dans la table *I*.

2.5.2 Attracteur

Certains états seront revisités souvent, cela dépend de l'état initial de départ. De tels états sont appelés attracteurs.

2.5.3 Bassin d'attraction

Le bassin associé à un attracteur est l'ensemble de tous les états qui mènent à cet attracteur. Dans la figure 9, les états (110-111) forment un attracteur et les autres états qui y induisent sont son bassin d'attraction. Les attracteurs représentent les points fixes (boucles) du système dynamique qui déterminent son comportement à long terme [1].

Les attracteurs sont toujours cycliques et peuvent avoir plus qu'un état. Le nombre de transitions pour retourner d'un état donné dans un attracteur est appelé la longueur de cycle [1] ou aussi longueur de l'attracteur.

Par exemple, l'attracteur (001) a une longueur de cycle égale à 1 tandis que les états 110-010 comprennent un attracteur avec une longueur de cycle égale à 2.

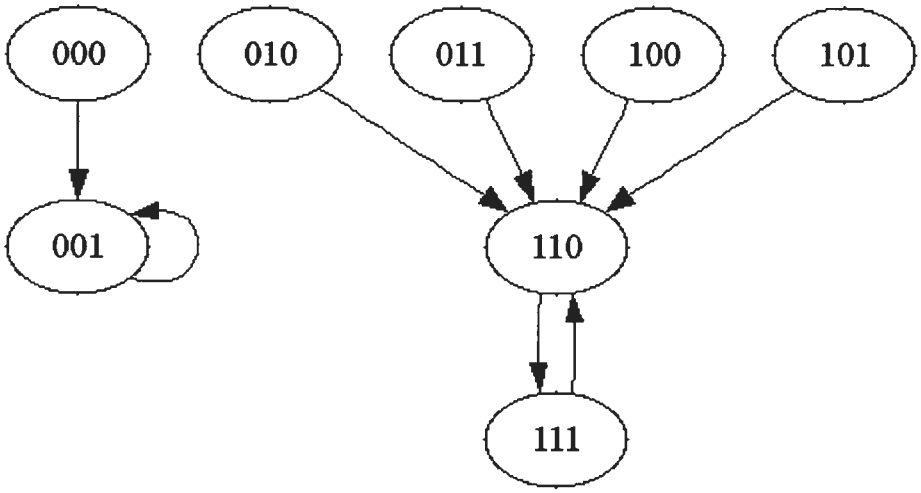


Figure 9 : Diagramme d'états pour un réseau booléen défini dans la table *I*

2.5.4 Dynamique dans les réseaux booléens

Un réseau booléen est déterministe, il suit une trajectoire unique à partir de l'état initial. Le seul aspect aléatoire qui peut exister dans les réseaux booléens est celui du choix de l'état initial du réseau en considérant la probabilité de distribution commune de tous les gènes. On commence d'abord par choisir l'état initial, ensuite on doit appliquer d'une façon répétitive les fonctions de transition (fonctions booléennes) à tous les gènes. Dans la première étape, les états initiaux sont utilisés comme des entrées pour les fonctions de transition pour produire de nouveaux états pour tous les gènes. À l'étape suivante les fonctions de transition sont alors appliquées aux nouveaux états et ainsi de suite. En effet, l'évolution dans le temps du réseau ne dépend plus des états initiaux mais des fonctions de transition.

Donc afin d'étudier le comportement dynamique à long terme de tels réseaux, nous devons considérer les probabilités communes de toutes les fonctions booléennes correspondantes à tous les nœuds [2].

Soit un réseau booléen *G(V, F)* composé de *n* nœuds (gènes) $x_1, x_2, ..., x_n$ et d'une distribution de probabilité jointe initiale $D(x)$, $x \in \{0, 1\}^n$.

On sait que dans un réseau booléen on associe à chaque nœud une seule fonction booléenne. La probabilité de sélection de la fonction booléenne associée à un nœud est égale à 1 (sinon 0).
On veut calculer la probabilité de chaque nœud après une étape du réseau pour pouvoir étudier le comportement du système à long terme. On peut écrire que :

$$\Pr\{f_1(x)=i_1, f_2(x)=i_2,\ldots,f_n(x)=i_n\} \qquad (2)$$
$$= \sum_{x\in\{0,1\}^n : f_k(x)=i_k, k=1,\cdots,n} D(x)$$

où $i_k \in \{0,1\}$, (2) peut être utilisée d'une façon itérative.

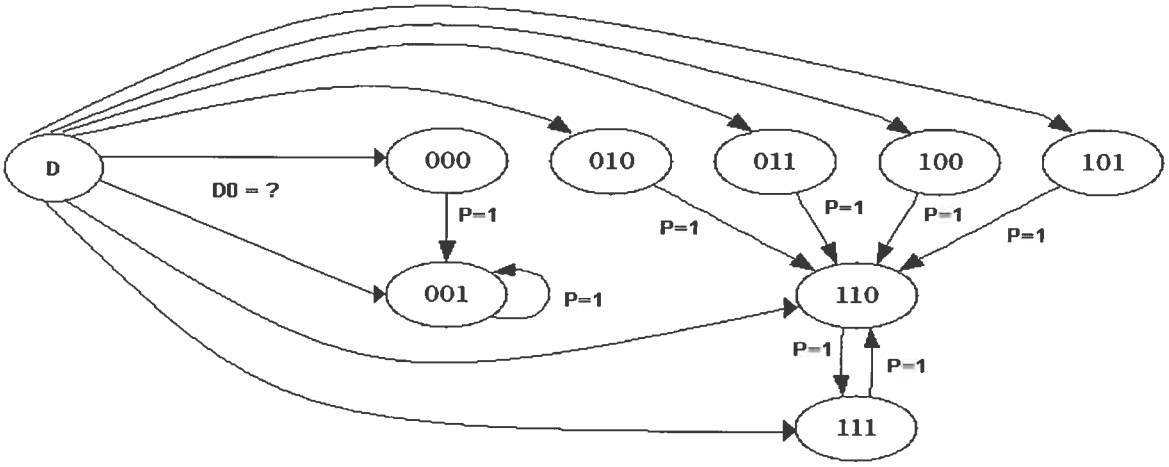


Figure 10 : Dynamique du réseau booléen

Puisque les nœuds seront dépendants mais indépendamment de *D(x)* après une étape du réseau, on peut remplacer *D(x) par* $\Pr\{f_k(x)=i_k, k=1,...,n\}$ dans (2) pour déterminer la distribution jointe à l'instant suivant.
L'état d'un nœud à l'instant t+1 ne dépend que du choix d'un nœud à l'instant t et non pas des nœuds aux instants antérieurs.
Ceci correspond exactement à la définition de chaîne de Markov [25]. Une chaîne de Markov est caractérisée par sa matrice de transition qui explicite les différentes transitions dans le réseau.

Dans le cas du réseau booléen, le contenu de la matrice de transition est déterminé par les fonctions de transition. Elles définissent l'état dans lequel le système devra se trouver à l'instant t en fonction de leurs entrées à l'instant t-1.

On définie la matrice de transition A de la manière suivante :

$$A_{ij} = \begin{cases} 1, & \exists x \in \{0,1\}^n,\ C(f_1(x),\ldots,f_n(x)) = j,\ C(x_1,\ldots,x_n) = i \\ 0, & \text{sinon} \end{cases}$$

où $C(i_1,\ldots,i_n) = 1 + \sum_{J=1}^{N} 2^{N-J} \cdot i_j$ et chaque $i_j \in \{0, 1\}$ tel que i et j sont de simples indices où i est la représentation entière du vecteur binaire $(x_1, x_2, \ldots, x_n)$ et j code le vecteur binaire $(f_1(x), f_2(x), \ldots, f_n(x))$ [2].

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} \text{1=000} \\ \text{2=001} \\ \text{3=010} \\ \text{4=011} \\ \text{5=100} \\ \text{6=101} \\ \text{7=110} \\ \text{8=111} \end{matrix}$$

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix}$$

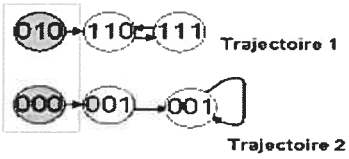


Figure 11 : Matrice de transition

La distribution de probabilité jointe à l'instant t+1 est complètement déterminée par la distribution de probabilité jointe à l'instant t et les probabilités de transition des états du réseau [25]. Ceci définit le système d'une façon itérative, on peut alors, écrire que :

$$D^{t+1} = \Psi(D^t) \qquad (3)$$

où l'application $\Psi : [0,1]^{2^n} \rightarrow [0,1]^{2^n}$ nous donne les probabilités de transition dans le réseau. On peut voir que la matrice A contient exactement une seule entrée non nulle dans chaque rangée. Donc on peut écrire (3) de la manière suivante [2, 25] :

$$D^{t+1} = D^t \cdot A = D^0 \cdot A^{t+1} \qquad (4)$$

où $D^0 = D(x)$ est la distribution de probabilité initiale.

2.5.5 Attracteurs et stabilité

Les réseaux booléens reflètent qualitativement la nature des systèmes adaptatifs complexes dans lesquels ils sont des systèmes composés d'agents qui interagissent en termes de règles. La stabilité structurelle est considérée dans ces systèmes comme une propriété essentielle reflétant le comportement du système face aux perturbations externes. C'est une propriété importante dans les réseaux de régulation génétiques réels puisque la cellule doit être toujours capable de maintenir son homéostasie[2] dans son programme de développement face aux perturbations externes [1, 4].

Les attracteurs dans les réseaux booléens représentent les états cellulaires comme la prolifération (cycle de la cellule), apoptose (mort de cellule programmée) et différentiation (exécution de tâches spécifiques au tissu) [5]. Beaucoup de stimuli différents (par exemple les radiations, chimiothérapie) peuvent mener au même état cellulaire [4, 5]. Ces états attracteurs sont stables sous des perturbations minimales [4, 5].

2.5.6 L'inférence

Pour comprendre la régulation génétique dans des organismes spécifiques et pour développer des outils pour l'intervention thérapeutique efficaces dans des maladies comme par exemple le cancer, il est nécessaire d'identifier les réseaux de régulation à partir des données expérimentales réelles. Beaucoup de travaux sur les réseaux booléens se sont concentrés sur l'identification de la structure du réseau à partir des données d'expression de gènes. Il existe plusieurs algorithmes d'inférence capables d'extraire les interactions génétiques à partir des données d'expression de gènes. Dans le cadre de ce mémoire, on présentera deux méthodes. La première méthode est basée sur l'utilisation des coefficients de détermination et la seconde méthode est appelée REVEAL, elle est basée sur l'utilisation de l'entropie de Shannon.

[2] L'homéostasie se définit comme la capacité de l'organisme de maintenir un état de stabilité relative des différentes composantes de son milieu interne et ce, malgré les changements constants de l'environnement externe.

2.5.7 Le coefficient de détermination

Une approche statistique générale est de découvrir des associations entre les différentes expressions de gènes via les coefficients de détermination (COD).

Ce coefficient mesure le degré avec lequel les niveaux d'expression d'un ensemble de gènes observés peuvent être utilisés pour prédire l'expression d'un gène cible [1, 2, 21, 22, 23].

Cela se fait en déterminant si l'information sur le niveau d'expression d'un petit ensemble de gènes peut être utilisée pour prédire d'autres gènes. La méthode permet l'incorporation de certaines connaissances qui relèvent de la prédiction comme par exemple la présence d'inactivant pour empêcher les mutations ou aussi l'application d'un stimulus particulier, ceci comme des éléments prédictifs affectant le niveau d'expression d'un gène donné [22, 23]. Dans cette méthode, il n'y a aucune supposition quant au mécanisme avec lequel les sorties sont déterminées et aussi aucune supposition sur la causalité de la méthode de prédiction [22, 23]. Le problème de prédiction est d'estimer les sorties d'un système seulement en considérant ses entrées. Cela en identifiant les ensembles de gènes qui peuvent être associés au gène cible.

Pour découvrir les relations prédictives dans l'ensemble des gènes, on laisse chaque gène être un gène cible et on conçoit des prédicteurs pour chacune des combinaisons de k $(1 \leq k \leq 3)$ [22, 23] gènes prédicteurs. Si le prédicteur conçu possède une erreur basse alors on peut dire que les gènes prédicteurs possèdent une relation avec le gène cible [21].

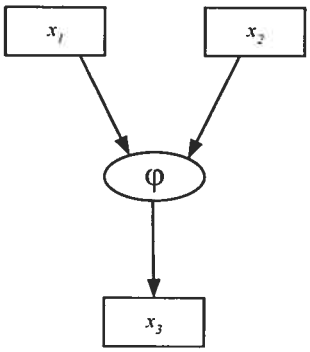
Un échantillon de N données est divisé d'une façon aléatoire en un ensemble composé de E données estimés et un ensemble formé de T=N-E données test. Le prédicteur est conçu à partir des E données estimées ensuite il est alors appliqué à l'autre ensemble de test pour obtenir une erreur de test qui sert comme une estimation de l'erreur de la population du prédicteur conçu [21].

L'algorithme commence d'abord par catégoriser les niveaux d'expression de gènes en données ternaires :-1 sous exprimé, 0 invariant et enfin 1 surexprimé [21, 22, 23].

En se basant sur les données de l'ensemble des E données estimées, la probabilité que le gène cible prenne un des trois états est calculée en prenant la probabilité conditionnelle de toutes les combinaisons des gènes prédicteurs.

La valeur du gène prédit est définie comme étant l'état ayant la plus grande probabilité conditionnelle [21]. Dans le réseau booléen, les expressions des gènes ont seulement deux

valeurs possibles 0 ou 1, alors on peut utiliser les fonctions booléennes pour modéliser les relations entre les gènes.

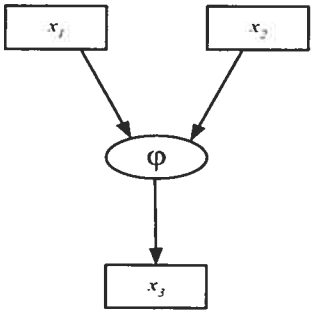


x_1	x_2	φ (x_1, x_2)
0	0	0
0	1	0
1	0	1
1	1	0

Un exemple d'une fonction booléenne

Figure 12 : Une fonction booléenne

Aussi, le comportement de x_3 peut être prédit par une fonction constante



x_1	x_2	φ = c
0	0	0
0	1	0
1	0	0
1	1	0

Un exemple d'une fonction constante

Figure 13: Une fonction constante

Dans le cas de la figure 13, x_3 ne dépend pas de x_1 et x_2 alors on peut écrire que $\phi = \phi_0 = c$ (l'index 0 dénote l'absence de prédicteurs). Chaque fonction booléenne (ou prédicteur) imite le comportement biologique. La qualité de cette imitation est mesurée par une mesure d'erreur. Il existe toujours une fonction binaire (prédicteur) optimale qui adapte le mieux le modèle biologique.

Parmi toutes les fonctions booléennes ϕ qui permettent de prédire le gène x_3, celle qui possède l'erreur minimale de mesure ε, est appelée la fonction optimale et notée ϕ_{opt}. ε est définie comme étant l'erreur moyenne quadratique (MSE).

Dans ce cas on a toujours : $\varepsilon[\phi_{opt}] \leq \varepsilon[\phi]$ pour n'importe quelle fonction booléenne ϕ.

De la même manière, le prédicteur optimal constant est appelé ϕ_{0-opt} (il n'y a que deux prédicteurs constants possibles 0 et 1). Si le gène x_3 est presque constant alors $\varepsilon[\phi_{0-opt}]$ doit être petit.

Le coefficient de détermination de la paire x_1 et x_2 comme prédicteurs du gène x_3 est donnée par l'amélioration relative dans la prédiction lorsqu'on utilise le prédicteur optimal ϕ_{opt} sur le prédicteur constant optimal ϕ_{0-opt} :

$$\theta = \frac{\varepsilon[\phi_{0-opt}] - \varepsilon[\phi_{opt}]}{\varepsilon[\phi_{0-opt}]}$$

Exemple : Essayons de déterminer le coefficient de détermination des gènes x_1, x_2 et x_3.

Table II : Données expérimentales binaires

	Exp_1	Exp_2	Exp_3	Exp_4	Exp_5	Exp_6	Exp_7	Exp_8
x_1	1	1	0	1	1	1	1	1
x_2	0	1	1	1	0	1	0	1
x_3	0	1	0	1	0	1	0	0

Divisons d'abord les données en deux ensembles

Table III : Données estimées

Estimation	Exp_1	Exp_2	Exp_3	Exp_4
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	1

Table IV : Données de test

Test	Exp_5	Exp_6	Exp_7	Exp_8
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	0

On doit estimer les fonctions optimales ϕ_{opt} et ϕ_{0-opt} pour $\{x_1, x_2\}$ comme un prédicteur pour le gène x_3.

Estimation de ϕ_{opt}

X dénote une configuation non observée,
elle prend la valeur la plus répandue,
on peut aussi utiliser une fonction heuristique

Estimation	Exp_1	Exp_2	Exp_3	Exp_4
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	1

Inférence statistique
de la fonction optimale
ϕ_{opt}

x_1	x_2	$\phi_{opt}(x_1, x_2)$	$\phi_{opt}(x_1, x_2))$
0	0	X	0
0	1	0	0
1	0	0	0
1	1	1	1

Test	Exp_5	Exp_6	Exp_7	Exp_8
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	0

On estime l'erreur de ϕ_{opt} à partir de l'ensemble Test

On remarque qu'il y a 1 erreur sur 4 alors $\varepsilon^*[\Phi_{opt}] = \mathbf{0,25}$

Estimation de ϕ_{0-opt}

Estimation	Exp_1	Exp_2	Exp_3	Exp_4
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	1

Inférence statistique
de la fonction optimale
ϕ_{0-opt}

La fréquence des valeurs
possible de x_3

x_1	fréquence
0	2
1	2

ϕ_{0-opt} = 1

on utilise une fonction heuristique
(la valeur la plus fréquemment
observée de x_3),

Test	Exp_5	Exp_6	Exp_7	Exp_8
x_1	1	1	0	1
x_2	0	1	1	1
x_3	0	1	0	0

On estime l'erreur de ϕ_{0-opt}
à partir de l'ensemble Test

On remarque qu'il y a 3 erreurs sur 4 alors $\varepsilon^*[\Phi_{o-opt}] = \mathbf{0,75}$

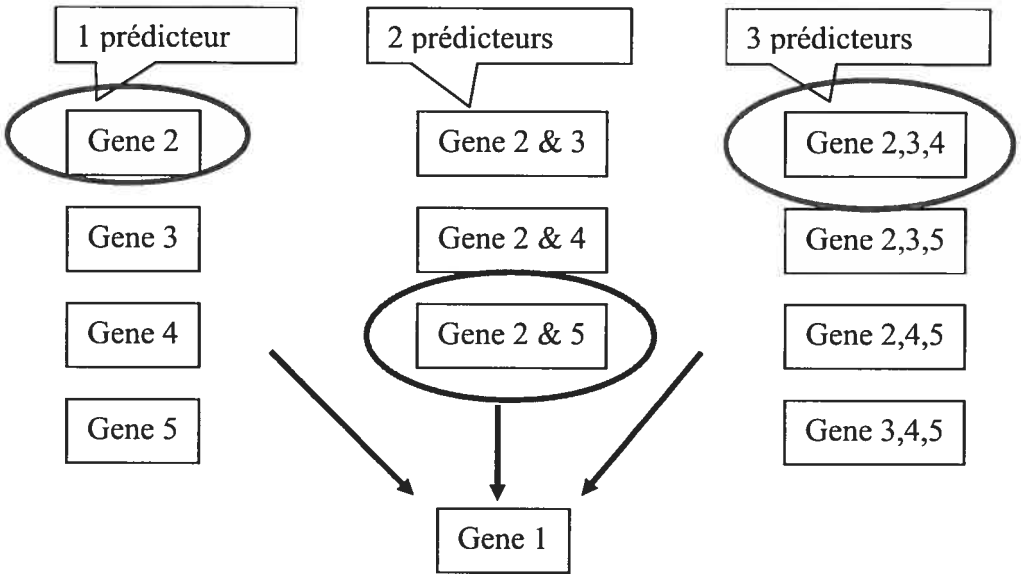
Donc le COD de x_1, x_2 et x_3 est donnée comme suit :

$$\theta^* = \frac{\varepsilon^*[\phi_{0-opt}] - \varepsilon^*[\phi_{opt}]}{\varepsilon^*[\phi_{0-opt}]} = \frac{0.75 - 0.25}{0.75} = 0.66$$

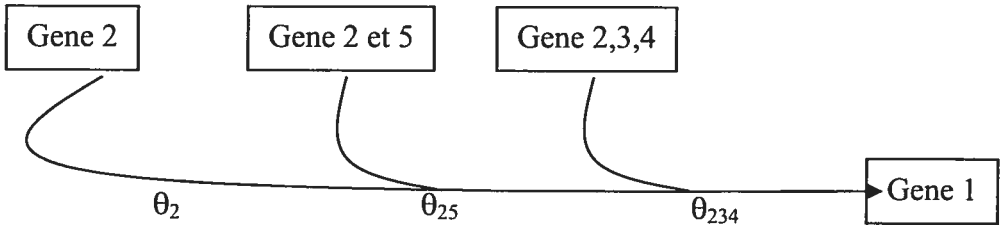
Le processus de division des N données en E données estimées et T données tests est répété d'une façon aléatoire 256 fois [22, 23]. La valeur estimée du COD est la moyenne de 256 valeurs de θ^*.

Méthodologie

On doit calculer le COD pour toutes les combinaisons de 1, 2 et 3 gènes prédicteurs pour chaque gène cible



Ainsi, on détermine les meilleurs ensembles de gènes prédicteurs probables pour chaque gène cible.



Ces ensembles de gènes (prédictifs) qui ont des COD les plus élevés comparativement aux autres ensembles de gènes, sont ceux employés pour construire le prédicteur optimal du gène cible.

2.5.8 REVEAL (Reverse Engineering Algorithm)

Cet algorithme a été développé par Liang, Fuhrman et Somogyi. REVEAL prend en entrée un ensemble de séries temporelles. Son but est de rechercher la connectivité entre les différents éléments du réseau puis de déterminer les règles logiques qui régissent les interactions génétiques, aussi de pouvoir prédire l'évolution du réseau dans le temps [6].

Pour cela, l'algorithme utilise la théorie de l'information notamment l'entropie de Shannon. De cette manière, on considère donc l'organisme comme un système qui peut fournir de l'information. Cette information a la forme de séries de 0 et de 1.

Pour trouver ces règles, il faut d'abord mesurer les niveaux d'expression de tous les gènes dans le réseau plusieurs fois pour chaque état. Ceci peut correspondre par exemple à l'exécution de plusieurs expériences et à prendre le niveau d'expression d'un gène dans plusieurs points. Ensuite, il faut rendre les différentes données ainsi récupérées sous forme binaire. La Figure 14 montre les expressions de trois gènes dans le temps, l'expression du gène i est considérée comme 1 ou « on » si elle est au delà du niveau τ_i , par contre elle sera considérée comme 0 ou « off » si elle se trouve au dessous de τ_i .

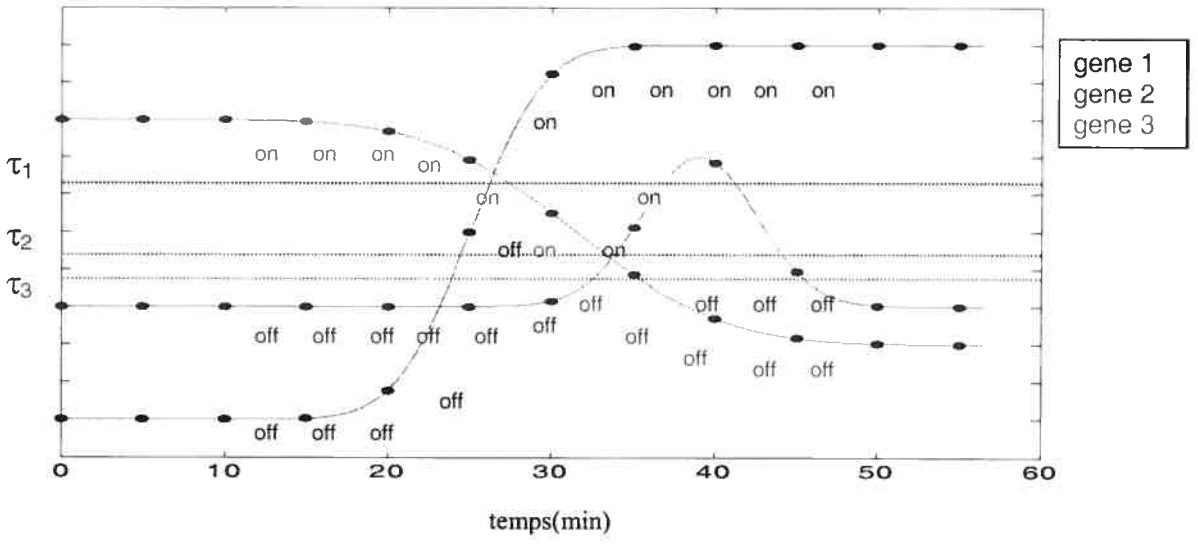


Figure 14 : Données d'expression de gènes temporelles

Nous obtenons alors les données binaires suivantes :

Table V : Expression de gènes sous forme binaire

Temps	0	5	10	15	20	25	30	35	40	45	50	55
gène 1	0	0	0	0	0	0	1	1	1	1	1	1
gène 2	0	0	0	0	0	0	0	1	1	0	0	0
gène 3	1	1	1	1	1	1	1	0	0	0	0	0

2.5.8.1 REVEAL et l'entropie de Shannon

Supposons qu'un gène x_1 a une valeur 0 à l'instant t, quelle est alors la probabilité que le gène x_2 ait la même valeur à l'instant $t+1$.

On se demande quel est le niveau d'incertitude qui entre dans la détermination de la valeur du gène x_2 une fois que l'on connaisse la valeur du gène x_1. Cela nous donne une mesure de l'influence de gène x_1 sur le gène x_2. Le rôle de l'incertitude impliquée dans un choix peut être évalué quantitativement par une mesure appelée l'entropie de Shannon.

Supposons que la variable aléatoire x prend la valeur i avec la probabilité p_i.

Alors, l'entropie de Shannon est définie en termes de probabilités en observant un symbole ou un événement p_i.

$$H(x) = -\sum_i p_i \log p_i \qquad \text{où } p_i = \text{probabilité}(x = i)$$

L'objectif de la méthode REVEAL, est de trouver les relations fonctionnelles qui existent entre les différents gènes du réseau.

Considérons un exemple où les séquences des gènes x et y sont données comme suit :

$x = [0, 1, 1, 1, 1, 1, 1, 0, 0, 0]$

$y = [0, 0, 0, 1, 1, 0, 0, 1, 1, 1]$

Alors, l'entropie de Shannon peut être déterminée comme suit :

$$H(x) = -\sum p_i \log_2(p_i)$$
$$= -(P(x=0).\log_2(P(x=0)) + P(x=1).\log_2(P(x=1)))$$
$$= -(0.4 \log_2(0.4) + 0.6 \log_2(0.6))$$
$$= 0.971$$
$$H(y) = -\sum p_i \log_2(p_i)$$
$$= -(0.5 \log_2(0.5) + 0.5 \log_2(0.5))$$
$$= 1.0$$

De la même manière, l'entropie de Shannon commune des variables x et y est définie comme suit :

$$H(x, y) = -\sum p_i \log_n(p_i)$$
$$= -(P(x=0, y=0).\log_2(P(x=0, y=0)) + P(x=0, y=1).\log_2(P(x=0, y=1))$$
$$+P(x=1, y=0).\log_2(P(x=1, y=0)) + P(x=1, y=1).\log_2(P(x=1, y=1)))$$
$$= -(0.1 \log_2(0.1) + 0.4 \log_2(0.4) + 0.3 \log_2(0.3) + 0.2 \log_2(0.2)$$
$$= 1.85$$

L'entropie conditionnelle : elle mesure l'incertitude dans la détermination de x une fois que l'on connaisse y.

$$H(x, y) = H(x|y) + H(y)$$
$$= H(y|x) + H(x)$$

L'information mutuelle : soit deux vecteurs binaires x et y, on définit l'information mutuelle portée par le couple (x, y) par la formule :

$$M(x, y) = H(y) - H(y|x)$$
$$= H(x) - H(x|y)$$
$$= H(x) + H(y) - H(x, y)$$

Cette quantité quantifie l'information que x et y portent en commun. Supposons que x soit une entrée et y soit une sortie, si $M(y, x) = H(y)$, alors x détermine y.

La stratégie de REVEAL est d'utiliser les mesures de l'information mutuelle pour extraire les relations qui existent entre les gènes à partir de la table de transition.

Il faut noter que pour N = 50 et avec une connectivité k <= 3 (entées par gène), un réseau complet ou correct peut être explicitement inféré à partir de tables de transition d'états incomplètes. En effet, pour N = 50 et k = 3, seulement 100 paires d'états de transitions sont nécessaires pour l'inférence d'un réseau booléen [6].

2.5.8.2 Étapes de la méthode REVEAL

Étape 1 : Un seul gène *x* détermine l'état du gène *y*

Rechercher les parties qui utilisent les règles d'entrées simples.

t	t+1
x	y
1	1
0	0

ou

t	t+1
x	y
1	0
0	1

Pour chaque y, tester chaque x_i et voir si $H(y, x_i) = H(x_i)$. Si un x_i est trouvé alors on a identifié le gène qui détermine complètement la sortie y. *À* ce moment il faut construire une table avec comme valeurs d'entrée x_i à l'instant t et valeurs de sortie y à l'instant (t+1) à partir de table de transition pour déterminer la règle logique qui gouverne la sortie y. Si par contre aucun x_i n'est trouvé alors la sortie y n'est pas déterminée par une seule entrée dans le réseau.

Étape2 : Deux gènes déterminent l'état du gène *y*

Toutes les combinaisons x_i, y_j sont testées comme entrées pour déterminer la sortie de chaque gène à qui on n'a pas assigné une règle logique d'une seule entrée.

Donc pour tous y (restant), on doit tester $H(y, x_i, x_j) = H(x_i, x_j)$ pour toutes les combinaisons *(i,j)* où $i \neq j$

Étape k : k gènes déterminent l'état du gène *y*

À l'étape k, tous les k tuples de gènes sont testés pour déterminer si

$H(y, x_1,x_2,\ldots,x_k) = H(x_1,x_2,\ldots,x_k)$.

Ce processus est répété jusqu'à ce qu'une règle pour chaque gène est déterminé ou le nombre d'entrées maximal est atteint.

Finalement, on doit déterminer l'information mutuelle entre les différents gènes pour déceler les rapports fonctionnels.

Prenons un exemple concret à partir de la table de transition donnée dans la table *VI*. Nous allons déterminer les règles logiques correspondantes en appliquant la méthode REVEAL.

Exemple : soit le réseau déterminé par sa table de transition :

Table *VI* : Table de transition.

Valeurs Entrées			Valeurs Sortie		
A_{i-1}	B_{i-1}	C_{i-1}	A_i	B_i	C_i
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1

D'abord, on doit calculer l'entropie de Shannon de chacun des trois gènes ainsi que les entropies communes entre les 3 gènes.

Calculer les entropies d'entrées

$H(A_{i-1}) = -((0.5)log(0.5) + (0.5)log(0.5)) = 1$

$H(B_{i-1}) = -((0.5)log(0.5) + (0.5)log(0.5)) = 1$

$H(C_{i-1}) = -((0.5)log(0.5) + (0.5)log(0.5)) = 1$

$H(A_{i-1}, B_{i-1}) = -((0.5)log(0.5) + (0.5)log(0.5) + (0.5)log(0.5) + (0.5)log(0.5)) = 2$

$H(B_{i-1}, C_{i-1}) = H(A_{i-1}, C_{i-1}) = 2$

$H(A_{i-1}, B_{i-1}, C_{i-1}) = 3$

Détermination des entrées du gène *A*

$H(A_i) = 1$

$H(A_i, A_{i-1}) = 2$ et $M(A_i, A_{i-1}) = 0.00$ alors $M(A_i, A_{i-1}) / H(A_i) = 0.00$

$H(A_i, B_{i-1}) = 1$ et $M(A_i, B_{i-1}) = 1.00$ alors $\mathbf{M(A_i, B_{i-1}) / H(A_i) = 1.00}$

$H(A_i, C_{i-1}) = 2$ et $M(A_i, C_{i-1}) = 0.00$ alors $M(A_i, C_{i-1}) / H(A_i) = 0.00$

Table *VII* : Table de la règle du gène A

entrée	sortie
B_{i-1}	A_i
0	0
1	1

$A_i = B_{i-1}$

Détermination des entrées du gène *B*

$H(B_i) = 0.81$

$H(B_i, A_{i-1}) = 1.50$ et $M(B_i, A_{i-1}) = 0.31$ alors $M(B_i, A_{i-1}) / H(B_i) = 0.38$

$H(B_i, B_{i-1}) = 1.81$ et $M(B_i, B_{i-1}) = 0.00$ alors $M(B_i, B_{i-1}) / H(B_i) = 0.00$

$H(B_i, C_{i-1}) = 1.50$ et $M(B_i, C_{i-1}) = 0.31$ alors $M(B_i, C_{i-1}) / H(B_i) = 0.38$

$H(B_i, [A_{i-1}, B_{i-1}]) = 2.50$ et $M(B_i, [A_{i-1}, B_{i-1}]) = 0.31$ alors $M(B_i, [A_{i-1}, B_{i-1}]) / H(B_i) = 0.38$

$H(B_i, [B_{i-1}, C_{i-1}]) = 2.50$ et $M(B_i, [B_{i-1}, C_{i-1}]) = 0.31$ alors $M(B_i, [B_{i-1}, C_{i-1}]) / H(B_i) = 0.38$

$H(B_i, [A_{i-1}, C_{i-1}]) = 2.00$ et $M(B_i, [A_{i-1}, C_{i-1}]) = 0.81$ alors $\mathbf{M(B_i, [A_{i-1}, C_{i-1}]) / H(B_i) = 1.00}$

Table *VIII* : Table de la règle du gène B

entrée		sortie
A_{i-1}	C_{i-1}	B_i
0	0	0
0	1	1
1	0	1
1	1	1

Donc, $B_i = A_{i-1}$ or C_{i-1}

Détermination des entrées du gène *C*

$H(C_i) = 1.00$

$H(C_i, A_{i-1}) = 1.81$ et $M(C_i, A_{i-1}) = 0.19$ alors $M(C_i, A_{i-1}) / H(C_i) = 0.19$

$H(C_i, B_{i-1}) = 1.81$ et $M(C_i, B_{i-1}) = 0.19$ alors $M(C_i, B_{i-1}) / H(C_i) = 0.19$

$H(C_i, C_{i-1}) = 1.81$ et $M(C_i, C_{i-1}) = 0.19$ alors $M(C_i, C_{i-1}) / H(C_i) = 0.19$

$H(C_i, [A_{i-1}, B_{i-1}]) = 2.50$ et $M(C_i, [A_{i-1}, B_{i-1}]) = 0.50$ alors $M(C_i, [A_{i-1}, B_{i-1}]) / H(C_i) = 0.50$

$H(C_i, [B_{i-1}, C_{i-1}]) = 2.50$ et $M(C_i, [B_{i-1}, C_{i-1}]) = 0.50$ alors $M(C_i, [B_{i-1}, C_{i-1}]) / H(C_i) = 0.50$

$H(C_i, [A_{i-1}, C_{i-1}]) = 2.50$ et $M(C_i, [A_{i-1}, C_{i-1}]) = 0.50$ alors $M(C_i, [A_{i-1}, C_{i-1}]) / H(C_i) = 0.50$

$H(C_i, [A_{i-1}, B_{i-1}, C_{i-1}]) = 3.00$ et $M(C_i, [A_{i-1}, B_{i-1}, C_{i-1}]) = 1.00$ alors

$M(C_i, [A_{i-1}, B_{i-1}, C_{i-1}]) / H(C_i) = 1.00$

Table *IX* : Table de la règle du gène C

entrée			sortie
A_{i-1}	B_{i-1}	C_{i-1}	C_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Donc on a: $C_i = (A_{i-1}$ and $B_{i-1})$ or $(B_{i-1}$ and $C_{i-1})$ or $(A_{i-1}$ and $C_{i-1})$

2.5.8.3 Difficultés avec l'implémentation de REVEAL

Il y a plusieurs difficultés avec l'implémentation actuelle de REVEAL. D'une part, il y a le problème de *k* (le nombre d'entrées par règle) qui peut augmenter considérablement. Le coût en matière de traitement pour la détermination des entropies de Shannon pour évaluer chaque *k-tuple* augmente d'une façon significative.

Le nombre de combinaisons qui doivent être testées dans chaque itération augmente considérablement. Si on a un réseau composé de n gènes, il y a C_n^k combinaisons à tester dans la k^{ieme} itération pour tout gène restant (n'ayant pas de règle logique).

Le choix de τ_i représente une autre difficulté pour l'implémentation de REVEAL, comment fixer τ_i pour affirmer qu'un niveau d'expression est 1 ou 0.

Réseau booléen probabiliste

Malgré la puissance de la logique booléenne qu'utilise le réseau booléen et la simplicité avec laquelle les interactions génétiques sont modélisées, il reste que ce modèle est déterministe [1]. Le fait de supposer une seule fonction logique par gène peut mener à des conclusions erronées. Dans le modèle booléen, les fonctions logiques sont inférées à partir de données expérimentales incertaines [1] leur exactitude n'est pas toujours assurée et ce pour divers raisons. D'abord, il y a l'incertitude biologique : l'expression de gène est en soi stochastique. Deuxièmement, il y a le bruit expérimental en raison du processus de mesure complexe.
Pour surmonter la rigidité des réseaux booléens, on a introduit une nouvelle classe de modèle appelée les réseaux booléens probabilistes (PBNs).

3.1 Réseaux booléens probabilistes (PBNs)

Ils sont des généralisations probabilistes des réseaux booléens standards. Ils offrent un cadre de modélisation flexible et puissant.
Les PBNs partagent les propriétés attirantes des réseaux booléens. Ils permettent l'étude systématique de la dynamique du réseau [2]. Cependant, à cause de leur nature probabiliste, ils sont capables de prendre en compte le non-déterminisme qui est intrinsèque aux systèmes biologiques [1, 2]. Les PBNs représentent une interface entre le déterminisme absolu des réseaux booléens et la nature probabiliste des réseaux bayésiens. L'idée fondamentale est de combiner plusieurs prédicteurs ou fonctions booléennes ensemble de sorte que chacun puisse apporter une contribution à la prévision d'un gène cible.
Les PBNs fournissent aussi une façon naturelle d'évaluer quantitativement l'influence relative et la sensibilité de gènes dans leurs interactions avec d'autres gènes [3]. Un des buts clefs de la modélisation de PBN est la détermination de cibles (gènes) possible pour l'intervention de sorte à ce que le réseau puisse être "forcé" à transiter dans un état désirable (ou ensemble d'états) [1, 2, 3].

3.2 Définition et Notation

Soit un ensemble de gènes $V = \{x_1, x_2, ..., x_n\}$, nous affectons à chaque x_i un ensemble $F_i = \{f_1^{(i)}, f_2^{(i)}, ..., f_{l(i)}^{(i)}\}$ de fonctions booléennes représentant les meilleurs prédicteurs pour ce gène cible.

Soit l le nombre de predicteurs possible pour un gène donné, si $l(i) = 1$ pour $i = 1,2,...,n$, alors le PBN est réduit simplement à un réseau booléen standard. Le bloc fonctionnel de base d'un PBN est montré dans Figure 15.

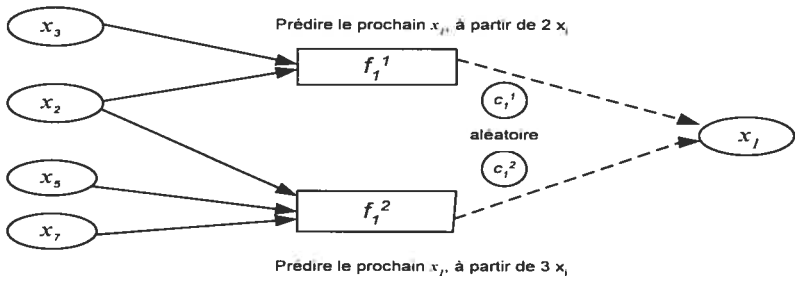


Figure 15 : Un gène et ses prédicteurs

Le prédicteur de chaque gène cible peut être considéré comme un commutateur aléatoire [1, 2] où à chaque instant ou étape du réseau, les $f_j^{(i)}$ sont choisies avec la probabilité $c_j^{(i)}$ pour prévoir le gène x_i.

De la même manière que pour les réseaux booléens, on peut utiliser le COD pour affecter les probabilités de sélection aux différents gènes du système tels que

$$\sum_{j=1}^{l_{(i)}} c_j^{(i)} = 1 \quad \text{, on a :} \quad c_k^{(i)} = \frac{\theta_k^i}{\sum_{j=1}^{l(i)} \theta_j^i} \qquad (5)$$

où θ_j^i est le COD pour le gène x_i relatif aux gènes utilisés comme entrées au predicteur $f_j^{(i)}$.

Un PBN est défini par, $V = \{x_1, x_2, ..., x_n\}$ un ensemble de nœuds et une liste $F = \{F_1, F_2, ..., F_n\}$ de fonctions booléennes où $F_i = \{f_1^{(i)}, f_2^{(i)}, ..., f_{l(i)}^{(i)}\}$.

Chaque nœud $\in \{0,1\}$ représente l'état (expression) du gène i .

$$x_i = \begin{cases} 1 & \text{signifie que le gène est exprimé} \\ 0 & \text{signifie que le gène n'est pas exprimé} \end{cases}$$

L'ensemble F_i représente les règles possibles des interactions de régulation pour le gène x_i, c'est-à-dire, que chaque $f_j^{(i)}$ $\{0,1\}^n \rightarrow \{0,1\}$ est une fonction booléenne déterminant la valeur du gène x_i en termes d'autres gènes et $l(i)$ est le nombre de fonctions possibles pour le gène x_i. Aussi, chaque noeud x_i transforme ses entrées en une sortie utilisant une règle logique $f_j^{(i)}$.

La mise à jour de tous les nœuds du système se fait d'une façon synchrone selon les fonctions qui leurs sont assignées. À un instant donné un des prédicteurs du gène x_i est choisi d'une façon aléatoire de l'ensemble F_i selon sa probabilité de sélection prédéfinie.

On peut facilement voir que l'espace d'état d'un PBN est identique à celui d'un réseau booléen standard. C'est-à-dire qu'il y a 2^n états possibles, chacun des états est représenté par combinaison binaire de longueur n.

La seule différence qui existe est que tandis que dans un réseau booléen standard les transitions sont déterministes (c-à-d., avec probabilité 1), dans un PBN, un état peut transiter à un certain nombre d'autres états selon une réalisation f_k qui est choisie à cet instant.

Une réalisation d'un PBN à un instant donné est déterminée par un vecteur de fonctions booléennes. S'il y a n réalisations possibles alors il y a n vecteurs de fonctions $f_1, f_2, ..., f_n$ de la forme $f_k = (f_{k_1}^{(1)}, f_{k_2}^{(2)}, ..., f_{k_n}^{(n)})$ pour $k = 1,...,n$ et $1<=k_i<=l(i)$ où $f_{k_i}^{(i)} \in F_i$ ($i = 1,...,n$). Le vecteur de fonctions f_k : $\{0,1\}^n \rightarrow \{0,1\}^n$ agit comme une fonction de transition qui représente une réalisation possible du PBN.

Soit $(x_1, x_2 ..., x_n)$ un ensemble de gènes et soit f_k l'ensemble des fonctions choisies pour la réalisation k, la valeur des gènes $(x_1, x_2 ..., x_n)$ après une étape du réseau est déterminée par

$f_k(x_1, x_2 ..., x_n) = (x'_1, x'_2, ..., x'_n)$ qui donne l'état des gènes après une étape du réseau par la réalisation f_k.

Soit $f = (f^{(1)}, f^{(2)}, ..., f^{(n)})$ un vecteur aléatoire prenant ses valeurs de $F_1 * F_2 * ... F_n$. C'est-à-dire que f peut prendre toutes les réalisations possibles du PBN, alors la probabilité que le predicteur $f_j^{(i)}$ soit utilisé pour prévoir le gène i ($1 <= j <= l(i)$) est égale à :

$$c_j^{(i)} = \Pr\{f^{(i)} = f_j^{(i)}\} = \sum_{k: f_k^{(i)} = f_j^{(i)}} \Pr\{f = f_k\} \qquad (6)$$

La dynamique des PBNs est identique à celle des réseaux booléens sauf que pour les PBNs, la valeur de chaque nœud à un instant donné est déterminée par un des prédicteurs possibles choisi selon sa probabilité de sélection. Une interprétation possible est qu'à tout moment nous avons un réseau actif sur les n réseaux possibles.

$$K = \begin{bmatrix} 1 & \cdots & 1 \\ 1 & \cdots & 2 \\ \vdots & \ddots & l(n) \\ l(1) & & l(n) \end{bmatrix}$$

Chaque rangée correspond à une configuration de réseau possible telle que la rangée r correspond au réseau r et l'entrée j dans la i^{eme} colonne spécifie le predicteur devant être utilisé pour prédire le gène x_i.

La probabilité que le réseau i soit sélectionné est :

$$P_i = \Pr\{Réseau\_i\_est\_selectionné\} = \prod_{j=1}^{n} c_{K_{ij}}^{(j)} \qquad (7)$$

où K_{ij} est la ij^{ieme} entrée dans la matrice K.

La probabilité pour que le réseau transite d'un état à un autre nombre d'autres états possibles peut être obtenu par :

$$A(x,x') = \Pr(\{(x_1,...,x_n) \rightarrow (x'_1,...,x'_n)\}$$

$$= \sum_{i=1}^{n} \Pr(\{(x_1,...,x_n) \rightarrow (x'_1,...,x'_n) | Réseau\_i\_est\_sélectioné\}).P_i \qquad (8)$$

3.3 Exemple

Pour illustrer ces notions, prenons l'exemple d'un PBN = *G (V, F)* où

$$V = \{x_1, x_2, x_3\} \qquad F_1 = \{f_1^{(1)}, f_2^{(1)}\}$$
$$F = \{F_1, F_2, F_3\} \qquad F_2 = \{f_1^{(2)}\}$$
$$F_3 = \{f_1^{(3)}, f_2^{(3)}\}$$

La table X représente les tables de vérité de tous les prédicteurs utilisés ainsi que leurs probabilités de sélection $c_j^{(i)}$.

Table X : Table de vérité

$x_1\ x_2\ x_3$	f_1^1	f_2^1	f_1^2	f_1^3	f_2^3
0 0 0	0	0	0	1	1
0 0 1	0	0	0	1	1
0 1 0	1	1	1	0	1
0 1 1	1	1	1	0	1
1 0 0	1	1	1	0	1
1 0 1	1	1	1	0	1
1 1 0	1	1	1	1	1
1 1 1	1	1	1	0	0
$c_j^{(i)}$	0.8	0.2	1	0.8	0.2

De cette table on peut construire les blocs de tous les gènes. La figure 16 montre le bloc de base du gène 1.

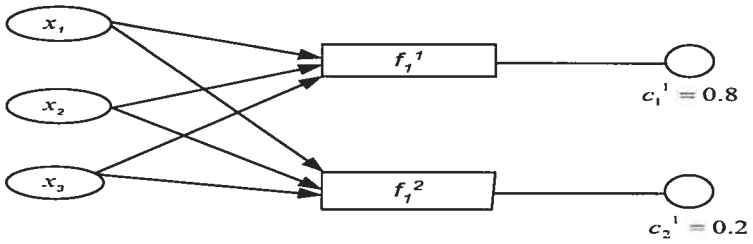


Figure 16 : Bloc de base du gène 1.

La construction de la table K se fait de la manière suivante : puisqu'il y a

- 2 fonctions pour le noeud x_1 $\quad F_1 = \{ f_1^{(1)}, f_2^{(1)} \}$
- 1 fonction pour le noeud x_2 $\quad F_2 = \{ f_1^{(2)} \}$
- 2 fonctions pour le noeud x_3 $\quad F_3 = \{ f_1^{(3)}, f_2^{(3)} \}$

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

#de genes

N = 4

Alors, le nombre de réseaux est $N = 2 * 1 * 2 = 4$

Construisons la matrice de transition A en prenant un cas concret. Considérons par exemple la probabilité de transition Pr{(1,0,0) -> (1,1,0)}.

Dans la table de vérité des fonctions, on peut voir qu'il y a deux combinaisons possibles.

$f_1^{(1)} = 1,\ f_1^{(2)} = 1,\ f_1^{(3)} = 0$ ou $f_2^{(1)} = 1, f_1^{(2)} = 1,\ f_1^{(3)} = 0$

Alors on peut prendre le chemin (1, 1, 1) ou (2, 1, 1) dans la table des réseaux K.

$x_1\, x_2\, x_3$	f_1^1	f_2^1	f_1^2	f_1^3	f_2^3
1 1 0	0	0	0	1	1

Les indices des chemins (1, 1, 1) et (2, 1, 1) de la matrice K sont respectivement 1 et 3.

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \\ 2 & 1 & 2 \end{bmatrix} \begin{matrix} \leftarrow 1 \\ 2 \\ \leftarrow 3 \\ 4 \end{matrix}$$

En utilisant (8), la probabilité de transition $A\ (5, 7)$ est égale à

$$\Pr\{(1, 0, 0) \longrightarrow (1, 1, 0)\} = P1 + P3.$$

100 est la 5ieme rangée

110 est la 7ieme colonne

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P1+P3 & P2+P4 \\ 0 & 0 & 0 & 0 & 0 & 0 & P1+P3 & P2+P4 \\ 0 & 0 & 0 & 0 & 0 & 0 & P1+P3 & P2+P4 \\ 0 & 0 & 0 & 0 & 0 & 0 & P1+P3 & P2+P4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

3.4 Dynamique des réseaux booléens probabilistes

Le comportement dynamique des PBN peut être représenté comme étant une chaîne de Markov où la matrice de transition d'états A est complètement spécifiée par toutes les fonctions booléennes et leurs probabilités. Supposons que dans l'exemple précédent, les probabilités des prédicteurs sont $c_1^{(1)} = 0.8,\ c_2^{(1)} = 0.2,\ c_1^{(2)} = 1,\ c_1^{(3)} = 0.8,\ c_2^{(3)} = 0.2.$

Alors d'après (7), on peut déterminer les probabilités des 4 réseaux.

Calculons par exemple la probabilité du premier réseau P_1 :

$$P_1 = c_1^{(1)} * c_1^{(2)} * c_1^{(3)} = 0.8 * 1 * 0.8 = 0.64$$

De la même manière, on détermine les probabilités des 4 réseaux qu'on peut substituer dans la matrice de transition A.

$$P_1 = 0.64,\ P_2 = 0.16,\ P_3 = 0.16,\ P_4 = 0.04.$$

Supposons maintenant que la distribution de probabilité jointe initiale est uniforme

$D^O = \left[\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\right]$, de ((4) : page 25) on peut déterminer les probabilités limites $\pi = [0, 0.25, 0, 0, 0, 0.525, 0.225]$, ceci indique qu'à long terme les trois gènes devront être 001 avec la probabilité de 0.25 et les trois gènes devront être 110 avec la probabilité de 0.525 par contre,

ils seront 111 avec une probabilité de 0.225. Ces trois états sont appelés des états absorbants. Cette notion correspond au concept des attracteurs dans le réseau booléen.

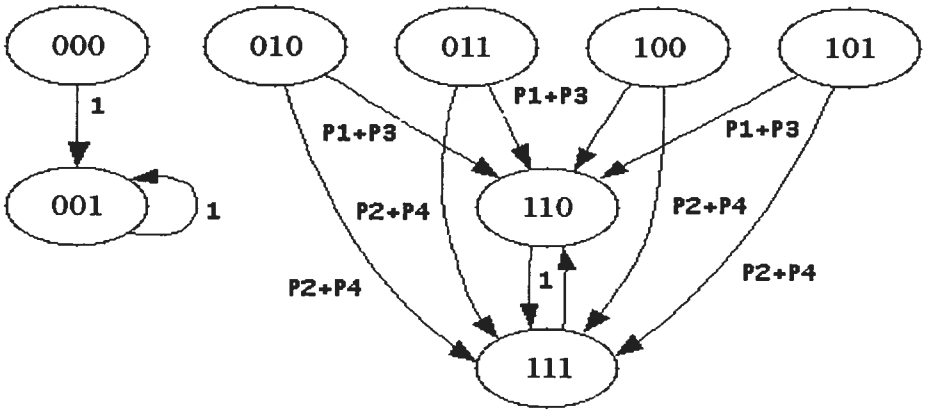


Figure 17 : Diagramme de transition d'états dans un PBN

3.5 Influence et sensibilité des gènes

L'un des points importants des PBNs, est leur pouvoir de déterminer les influences qui existent entre les différents gènes du système. On sait qu'il y a toujours des gènes qui sont plus importants que d'autres dans la détermination de la valeur du gène cible.

Alors, Il est important de distinguer ces gènes qui ont un impact important sur le prédicteur de ceux qui ont seulement un impact mineur. Dans une fonction booléenne, il existe des variables qui ont une puissance déterminative plus importante sur le résultat que d'autres.

Par exemple

$$f(x_1, x_2, x_3, x_4) = x_1 \vee x_2.x_3.x_4$$

f est utilisée comme prédicteur où le symbole « $\vee$ » représente la disjonction et le « . » représente la conjonction. On remarque que x_1 est plus important que les autres gènes parce que si on le met à 1, il forcera la fonction f à prendre la valeur 1 indépendamment de la valeur des autres variables. Par contre si la valeur de $x_1 = 0$, alors les autres variables doivent coopérer pour déterminer la valeur de la fonction f. Les influences des variables dans la fonction peuvent quantifier cette notion d'importance.

L'influence est définie en terme de dérivée partielle de la fonction booléenne *f*.

$$\frac{\partial f(x)}{\partial x_j} = | f(x) - f(x^{(j)}) | \equiv \frac{\partial f(x)}{\partial x_j} = f(x^{(j,0)}) \oplus f(x^{(j,1)}) \qquad (9)$$

On peut déterminer l'influence d'une variable x_i sur une fonction f de la manière suivante :

$$I_j(f) = E_D\left[\frac{\partial f(x)}{\partial x_j}\right] = \Pr\left\{\frac{\partial f(x)}{\partial x_j} = 1\right\} = \Pr\{f(x) \neq f(x^{(j)})\} \qquad (10)$$

L'influence totale du gène x_k sur le gène x_i peut être obtenu par :

$$I_k(x_i) = \sum_{j=1}^{l(i)} I_k(f_j^{(i)}) \cdot c_j^{(i)} \qquad (11)$$

La matrice des influences est donnée par $\Gamma_{ij} = I_i(x_j)$, elle contient les influences entre chaque paire de gènes.

Biologiquement, la sensibilité d'un gène représente sa stabilité ou aussi son autonomie [1, 3]. Si la sensibilité d'un gène est basse, ceci implique que d'autres gènes ont peu d'effets sur lui. Les PBNs nous permettent de calculer les influences entre gènes (ou ensembles de gènes).

Les gènes avec un facteur d'influence élevé peuvent avoir un impact collectif élevé sur les autres gènes. C'est justement ces gènes qui ont le potentiel déterminatif pour réguler le comportement dynamique du réseau et leur perturbation peut amener à des situations où le réseau ou un sous réseau peut être affecté. Possiblement que le système sera forcé à transiter à autre bassin d'attraction [3]. Ces gènes peuvent faire de bonnes cibles potentielles pour l'intervention.

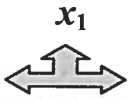
D'une façon analogue, on peut définir la sensibilité d'un gène comme étant la somme des influences agissant sur lui.

$$r(x_i) = \sum_{k=1}^{n} \Gamma_{ik} \qquad (12)$$

On peut facilement généraliser les notions d'influence et de sensibilité aux ensembles de gènes, c'est-à-dire qu'on peut définir l'influence d'un ensemble de gènes sur un autre ensemble de gènes.

3.5.1 Exemple de l'influence des gènes dans un PBN : Supposons qu'on veut déterminer l'influence de x_2 sur x_1.

Prenons l'exemple précédent, on doit d'abord calculer l'influence de x_2 sur les deux prédicteurs f_1 et f_2



$x_1\,x_2\,x_3$	f_1^1	f_2^1	f_1^2	f_1^3	f_2^3
0 0 0	0	0	0	1	1
0 0 1	0	0	0	1	1
0 1 0	1	1	1	0	1
0 1 1	1	1	1	0	1
1 0 0	1	1	1	0	1
1 0 1	1	1	1	0	1
1 1 0	1	1	1	1	1
1 1 1	1	1	1	0	0
$c_j^{(i)}$	0.8	0.2	1	0.8	0.2

Dans la table de vérité de l'exemple ci-dessus, on a que

$$f_1^{(1)}(x^{(2,0)}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad \text{et} \qquad f_1^{(1)}(x^{(2,1)}) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Utilisant ((9) : page 48), pour déterminer l'influence de la variable x_2 sur le prédicteur f_1.

$$I_2(f_1^{(1)}) = E_D\left[f_1^{(1)}(x^{(2,0)}) \oplus f_1^{(1)}(x^{(2,1)})\right]$$

Donc,

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

donc, (1+1) /4 = **0.5**

Donc, l'influence de x_2 sur $f_1^{(1)}$ est égale à : $I_2(f_1^{(1)}) = E_D\left[\dfrac{\partial f_1^{(1)}(x)}{\partial x_2}\right] = 0.5$

De la même manière, on peut calculer les autres influences

L'influence de x_2 sur $f_2^{(l)}$ est égale à : $I_2(f_2^{(1)}) = E_D\left[\frac{\partial f_2^{(1)}(x)}{\partial x_2}\right] = 0.75$

La somme des predicteurs du gène x_1 consiste en $f_1^{(1)}$ et $f_2^{(1)}$, donc on peut obtenir $I_2(x_1) = 0.8*0.6+0.75*0.2=0.6$. Enfin, on peut construire la matrice $I_j(x_i)$

De

x_1	0.1	0.75	0.375	=> $r(x_1)$
x_2	0.6	0.75	0.375	=> $r(x_2)$
x_3	0.6	0.75	0.375	=> $r(x_3)$

À x_1 x_2 x_3

On peut aussi avoir l'effet collectif d'un gène sur les autres gènes => $r(x_i)$

3.6 Intervention

La plupart des réseaux génétiques sont stables dans le sens ou ils fonctionnent dans des états qui sont stables aux perturbations. Dans les réseaux Booléens, cela correspond à un retour probable aux attracteurs, dans les PBNs il correspond à une sensibilité basse des probabilités stationnaires [3].

La modélisation PBN offre aux chercheurs l'outil pour déterminer les gènes qui ont un impact important sur le comportement du réseau. Ces gènes peuvent forcer le réseau à transiter dans un ou ensembles d'états désirés.

La perturbation de ces gènes est plus souhaitable afin de réaliser le résultat désiré [3]. Le but des chercheurs est donc de découvrir, quel gène est le meilleur dans le sens où sa perturbation permet d'obtenir le comportement du réseau recherché.

3.7 Solution basée sur l'utilisation des FPGAs

Les différents formalismes discutés dans le chapitre 2 permettent de modéliser les interactions génétiques de différentes façons. Certaines sont statiques (représentation graphique, réseaux bayésiens) alors que d'autres sont dynamiques (réseaux booléens, équations différentielles). Il y a des modèles qui sont déterministes (réseaux booléen) et d'autres prennent en compte le non déterminisme (réseaux probabilistes booléens). Cependant, pour tous ces formalismes le problème dit « problème de dimension » ou taille du système limite la représentation statique des réseaux de régulation.

Les solutions logicielles représentant d'une façon statique les réseaux de régulation génétique ont vu leurs limites atteintes (à l'exception des petits réseaux). En effet, de telles solutions ne peuvent manipuler de grands réseaux de régulation (au moins une centaine de gènes) en raison des contraintes sur la mémoire.

Cependant, une solution basée sur l'approche de conception conjointe matériel-logiciel permet de répondre aux attentes des chercheurs. Cette solution est capable de calculer des milliards d'équations logiques par seconde. Elle est aussi capable de modéliser un réseau de régulation génétique de taille quelconque (elle ne construit pas d'une façon explicite la table représentant le réseau) et surtout elle n'est pas coûteuse. C'est une solution très efficace avec des résultats impressionnants utilisant une plateforme dans laquelle cohabite un processeur ARM et un FPGA. Le recours à la technologie FPGA nous a permis de manipuler des réseaux de régulation génétiques de grande taille avec une puissance de calcul très impressionnante.

Pour montrer les raisons qui nous ont poussés à utiliser cette technologie des FPGAs, prenons un exemple où on a un réseau composé de 1000 gènes ayant chacun deux prédicteurs. La fréquence de notre FPGA est de 25 Mhz. Comme la détermination des états du réseau se fait d'une façon parallèle, ce FPGA peut calculer 25 millions d'états chaque seconde (on utilise des équations logiques simples). Puisqu'on a 1000 gènes, le FPGA nous permet de calculer 25 millions * 1000 équations logiques soit 25 milliards chaque seconde. Comme on a 2 prédicteurs par gène alors ce FPGA est capable d'atteindre le chiffre de 50 milliards équations logiques par seconde. Il faut noter que le FPGA utilisé est de taille moyenne, c'est un Virtex 2000 du fabricant Xilinx.

Avant de donner une description détaillée de notre solution, définissons d'abord c'est qu'est un circuit FPGA, comment on le programme et surtout quels sont ses avantages ?

3.8 FPGA (field programmable gate arrays)

3.8.1 Définition

Les FPGA (Field Programmable Gate Arrays) ou aussi, "réseaux logiques programmables" sont des composants entièrement re-configurables. L'avantage de ce genre de circuit est sa grande souplesse qui permet de le réutiliser pour réaliser différents systèmes numériques en un temps très court. Les circuits FPGAs sont constitués d'une matrice de blocs logiques programmables ou CLB (configurable logic bloc) entourés de blocs d'entrée et sortie programmables ou IOB (input output bloc). L'ensemble est relié par un réseau d'interconnexions programmables [15].

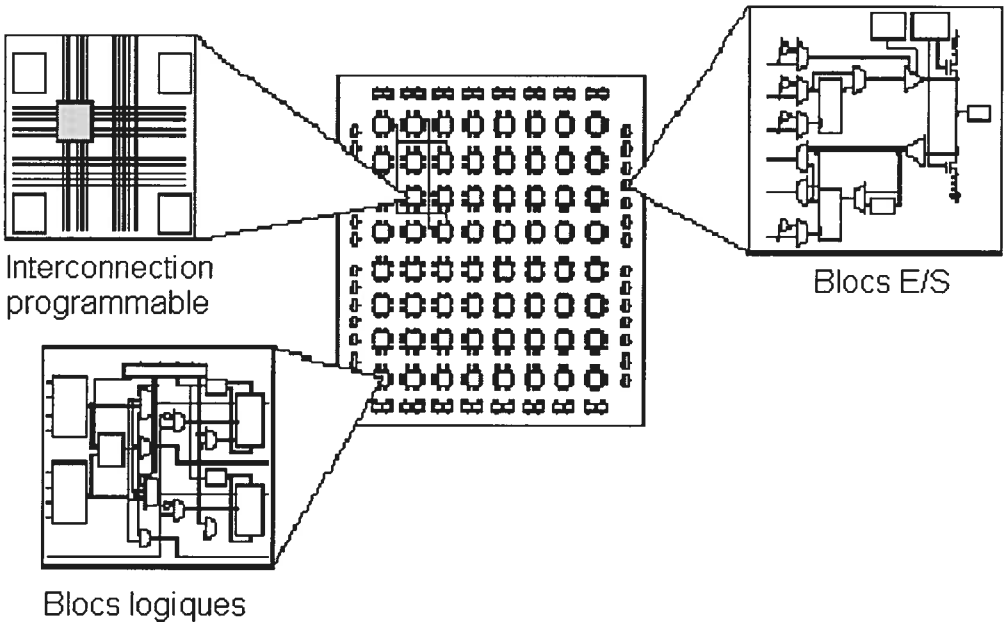


Figure 18 : Architecture interne d'un FPGA

L'architecture d'un bloc logique varie d'un FPGA à un autre ; elle peut être une porte logique NAND (le complément du « ET » logique) à deux entrées ou plus complexe comme un multiplexeur ou une LUT (Lookup-Table).

La plupart des blocs logiques contiennent des bascules de façon à permettre l'implémentation de circuits séquentiels. Les CLB permettent de réaliser des fonctions combinatoires et des fonctions séquentielles.

Un circuit logique est implémenté dans un FPGA en partitionnant la logique sur différents blocs logiques et en interconnectant ces différents blocs à l'aide des interrupteurs [27]. L'architecture des FPGAs du fabricant Xilinx est basée sur l'utilisation des LUTs.

LUT ("Look Up Table") : chaque cellule logique dispose de quatre entrées. Il existe donc $2^4 = 16$ combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits (la LUT). La LUT permet d'implémenter n'importe quelle fonction logique de 4 entrées [27].

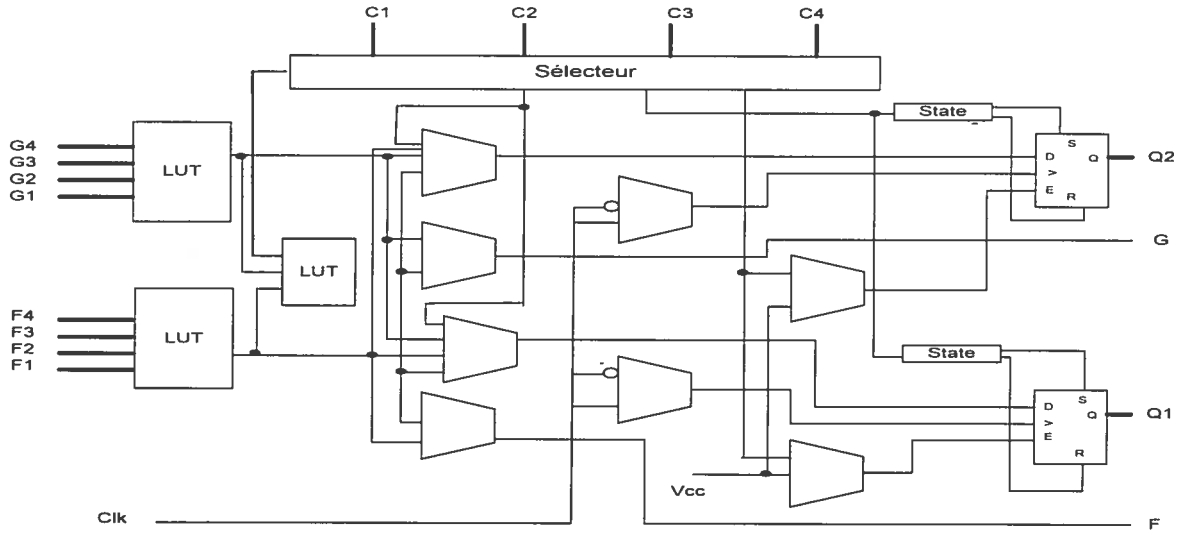


Figure 19 : Unité fonctionnelle d'un XC4000 (Xilinx)

3.8.2 Programmation d'un FPGA

La figure 20 décrit les différentes étapes nécessaires à la programmation d'un FPGA.

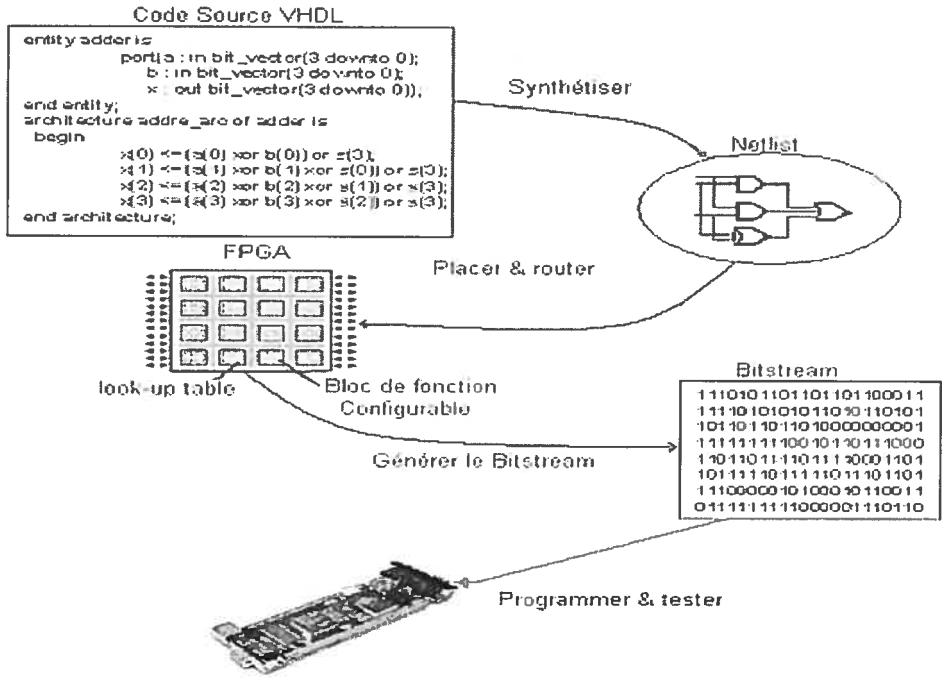


Figure 20 : Programmation d'un FPGA.

3.8.2.1 Spécification

On décrit la fonctionnalité du design à l'aide d'un langage de description de matériel comme par exemple VHDL ou VERILOG. Toutes les fonctionnalités du design peuvent être simulées en vue de leur validation.

3.8.2.2 Synthèse

L'outil de synthèse transforme automatiquement un programme écrit dans un langage de description de matériel en une interconnexion de circuits logiques [15]. Chacun de ces circuits est une fonction matérielle élémentaire du FPGA.

3.8.2.3 Placement / routage

Dans cette étape chaque fonction logique est placée dans une cellule logique du FPGA. L'outil de placement et routage connecte les différentes cellules logique du FPGA entre elles d'une part et avec les ports d'entrées et sortie d'autres part en respectant la fonctionnalité du design.

3.8.2.4 Génération du fichier de configuration

Cette étape consiste à générer un fichier de configuration appelé Bitstream. Ce dernier contient les informations fournies au composant FPGA (exemple : Virtex-2000E de Xilinx) afin qu'il prenne la configuration souhaitée.

3.8.2.5 Programmation et test

Le fichier Bitstream peut maintenant être envoyé au FPGA pour être programmé. Dès lors le FPGA se comporte comme le circuit décrit en 2.8.2.1.

3.8.3 Avantages des FPGAs

Les FPGA sont des composants complètement re-configurables. Ils contribuent d'une façon considérable à réduire le temps et le coût de conception des systèmes numériques. La taille des FPGA ne cesse d'augmenter, on parle de cent millions de portes logiques actuellement ce qui rend possible l'intégration des systèmes numériques complexes pour un coût moindre. À la différence d'un circuit ASIC, un FPGA peut être reprogrammée même après qu'il ait été utilisé dans un système numérique. Un FPGA peut être utilisé plusieurs fois pour concevoir différents designs.

Implémentation

4.1 Approche

Comme vu dans l'introduction, le graphe de transition d'états a un nombre exponentiel de noeuds comparativement au nombre de gènes considérés dans le système, la construction statique d'une table représentant un tel graphe peut devenir pratiquement impossible (problème de mémoire). Pour résoudre ce problème, on présente une solution efficace utilisant l'approche de co-design matériel-logiciel basée sur l'utilisation des FPGAs. Notre but est de détecter les attracteurs afin de connaître les états stables du système. Cela nécessite plusieurs tâches : la simulation du PBN (qui implique la génération de nombres aléatoires), la détection de cycles et leurs sauvegarde et finalement l'analyse de ces cycles pour trouver les attracteurs.
Les FPGA sont très puissants pour simuler un réseau booléen parce qu'ils sont configurables au niveau portes logiques et tout le réseau peut être vu comme étant une énorme fonction combinatoire tandis que les registres sont utilisés pour sauvegarder l'état courant de chaque gène. Un cycle horloge est suffisant pour exécuter toutes les équations logiques en parallèle et avoir l'état suivant de chaque gène. En plus de la simulation du PBN, le FPGA sera aussi utilisé pour détecter les cycles parce que cette tâche exige un nombre très important de comparaisons parallèles qui correspond exactement au contexte où le FPGA est puissant. Une vue globale de notre architecture est présentée dans la figure 21.

Le FPGA envoie les cycles au processeur d'une façon parallèle grâce à une interface FIFO.
Le processeur sauvegarde et trie les cycles tandis que le FPGA essaie de trouver un nouveau cycle. La FIFO est utilisée comme un tampon entre le FPGA et le processeur permettant ainsi de réguler l'échange.

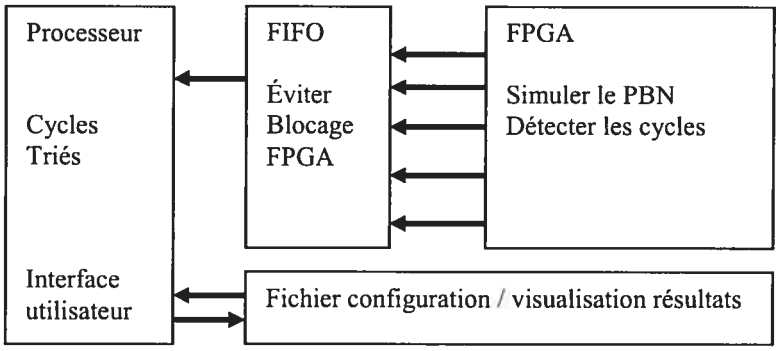


Figure 21 : Architecture du système.

Nous avons réalisé un système matériel-logiciel capable de simuler un réseau de régulation de gènes utilisant le PBN comme modèle.

Le système ainsi généré permet de réaliser les fonctions suivantes :

- La Simulation d'un réseau PBN
- L'identification et la sauvegarde des attracteurs.

Le système entier a été implémenté dans une plateforme « the rapid prototyping platform (RPP) » fournie par la Société canadienne de Microélectronique (CMC) [11, 12, 13].

4.2 Plateforme de prototypage rapide

La société canadienne de microélectronique a fourni une plateforme (RPP: Rapid Prototyping platform) aux universités pour la recherche. Elle se compose de composants matériels et logiciels. Elle permet le prototypage et la conception de système embarquée complexe autour d'un microprocesseur ARM7TDMI [10, 11, 12].

La RPP dispose de deux cartes (soeurs), les deux sont logées sur la même carte mère (ARM's Integrator/AP board). La RPP supporte le module ARM7TDMI et le module logique FPGA Virtex-2000E de Xilinx figure 22.

En plus de la génération d'horloge, l'arbitrage de bus et le système d'interruption, l'Intégrateur/AP fournit aussi un système d'exploitation (placé dans une mémoire flash) et des ressources d'entrée-sortie.

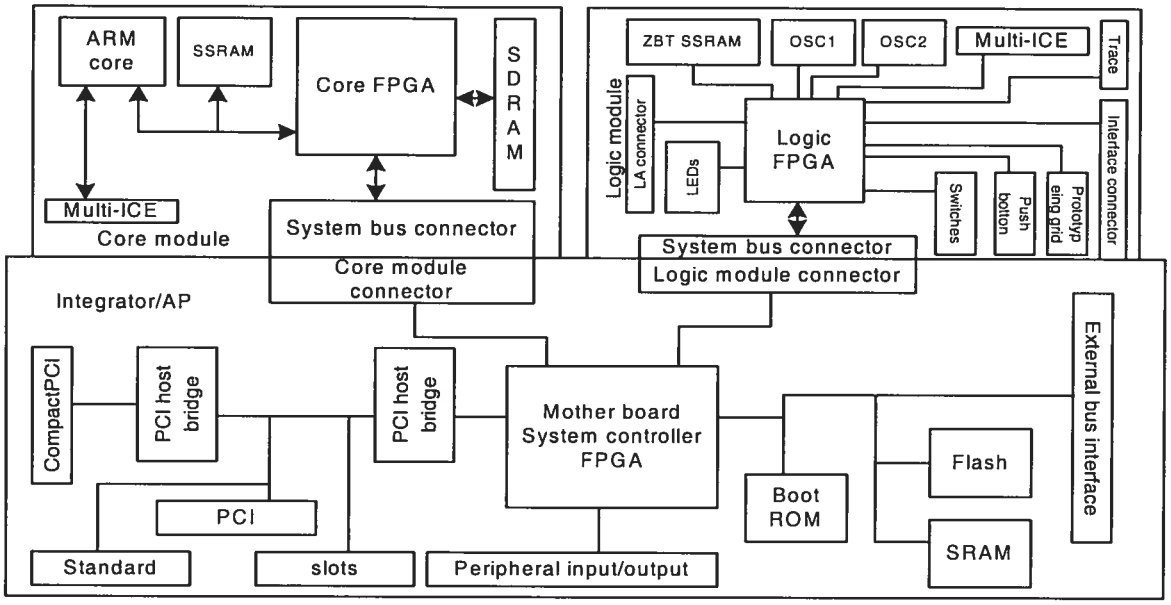


Figure 22 : Plateforme CMC

4.3 Architecture du système

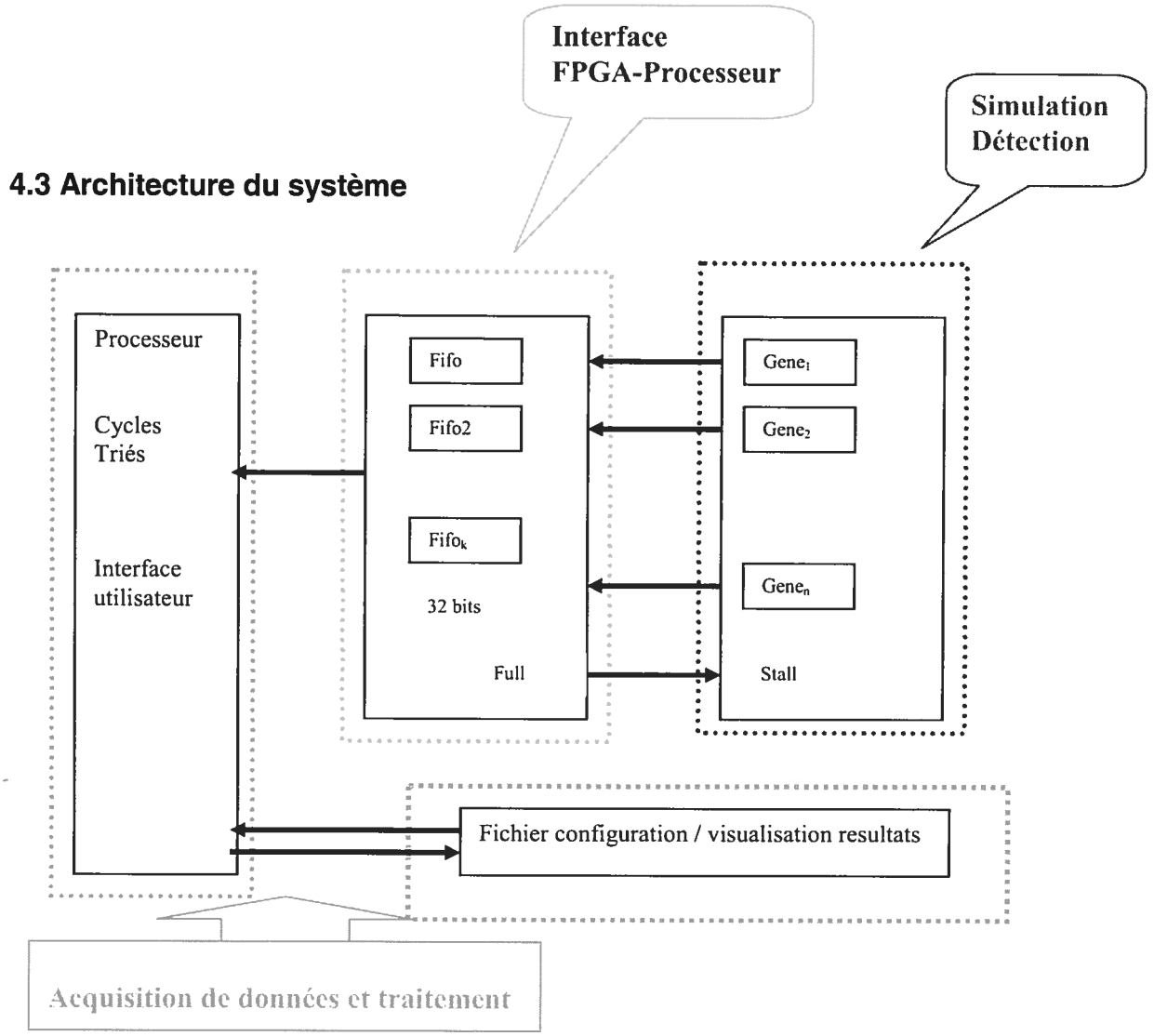


Figure 23 : Partitionnement du système

L'architecture de notre système (figure 23) montre qu'il est partitionné en trois parties. La première partie permet la simulation et la détection des cycles. La deuxième joue l'interface entre le FPGA et le processeur. Elle s'occupe aussi de l'écriture des cycles ainsi produits dans une ou plusieurs FIFO(s). Finalement, la troisième partie permet l'acquisition des données permettant la génération des fichiers VHDL de notre système. Elle permet aussi, la lecture et la sauvegarde des cycles et elle permet de déterminer le nombre d'occurrences de chaque cycle. Stall et Full sont des signaux utilisés pour synchroniser la partie 1 avec la partie 3 du système. Leur utilisation sera vue plus en détail dans les sections suivantes.

4.4 Partie matérielle

4.4.1 L'entité Gène

Chaque gène est composé d'un état (exprimé - non exprimé), un ensemble de prédicteurs qui sont des candidats pour calculer l'état suivant et une boîte de décision pour choisir d'une façon aléatoire un seul prédicteur (Figure 24).

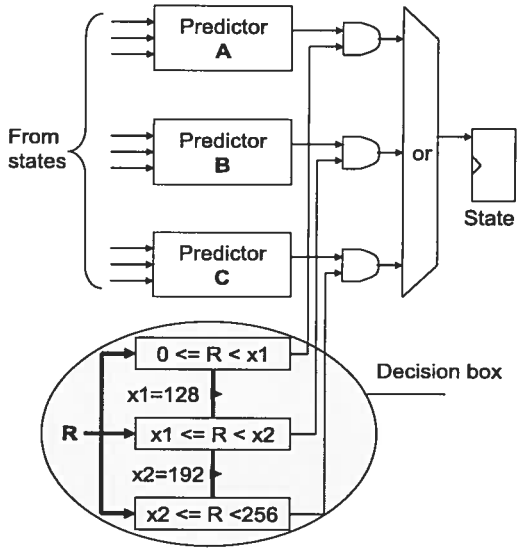


Figure 24 : Implémentation d'un gène

La décision est implémentée par une cascade de comparateurs qui divise l'espace en régions proportionnelles à la probabilité des prédicteurs connectés. Une valeur pseudo aléatoire (R) est utilisée pour choisir une région qui nous donne lequel des prédicteurs à utiliser. La figure 24 montre un cas où le premier prédicteur a une probabilité de ½ (la région 0-127) et les deux autres ont une probabilité de ¼ (des régions 128-191 et 192-255).

Examinons maintenant comment l'entité gène est réellement implémentée.

Pour chaque gène i, on lui correspond un ensemble $F_i = \{f_j^{(i)}\}\ j = 1, \ldots\ldots, l(i)$ où chaque $f_j^{(i)}$ est une fonction possible dans la détermination de la valeur du gène x_i et $l(i)$ est le nombre de fonctions possibles pour le gène x_i, on appelle aussi $f_j^{(i)}$ un prédicteur.

Chaque fonction a un nombre d'entrées quelconque qui peut être différent des autres fonctions du même gène x_i. Les variables d'entrées de chaque fonction peuvent être différents.

Un générateur de nombre aléatoire a été implémenté. Il permet de choisir lequel des prédicteurs sera utilisé pour prédire le gène x_i. Selon les entrées du predicteur choisi, la valeur du gène cible x_i sera sélectionnée et mise dans la sortie appropriée.

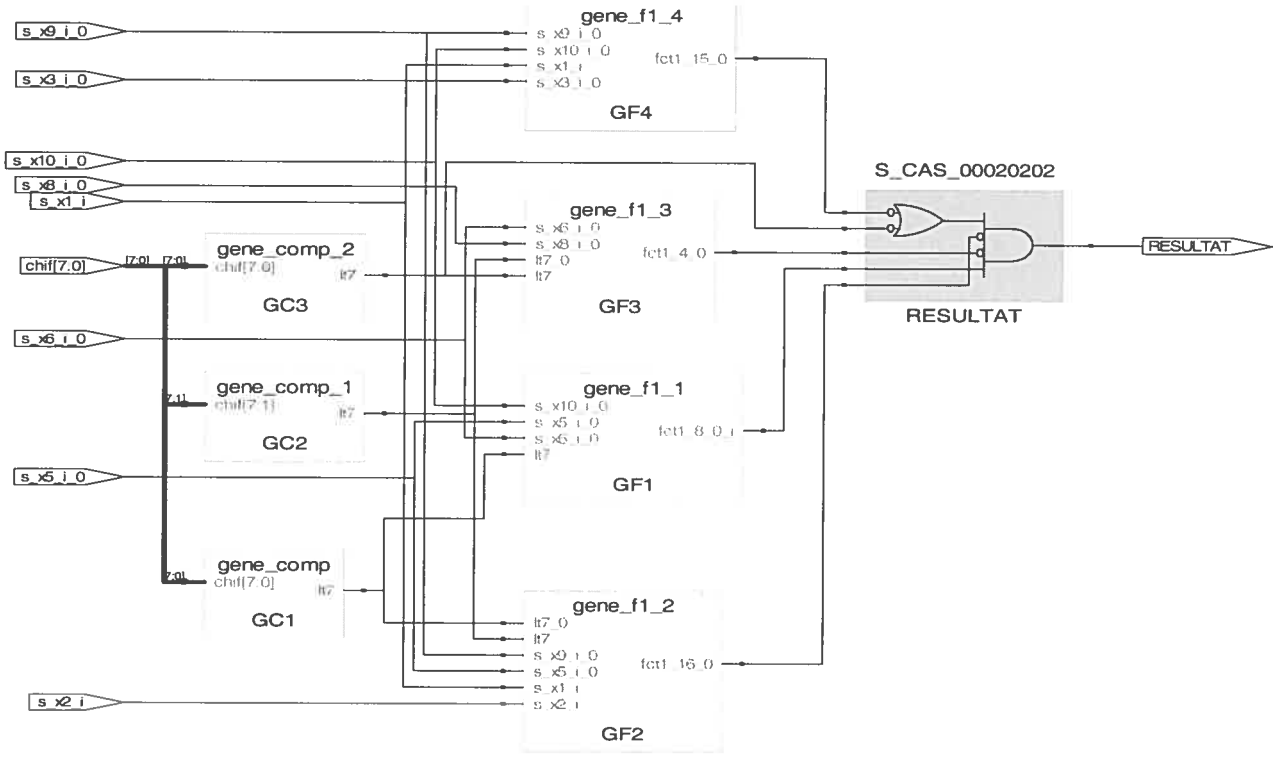


Figure 25: Un gène

La figure 25 montre l'implémentation matérielle d'un gène ayant 4 fonctions logiques et 3 comparateurs générée par l'outil de synthèse Synplify.

4.4.1.1 L'entité Gene_comp

C'est un comparateur, leur nombre diffère d'un gène à un autre. Il dépend du nombre de fonctions logiques associées à chaque gène.

Entrées de l'entité :

- RND
- NB

La fonction principale de cette entité est de comparer la valeur qui lui est associé NB (cette valeur représente la probabilité de sélection c_j^i) avec celle du nombre aléatoire RND associé au gène x_i. Elle permet de choisir la fonction logique qui lui est associée. Le choix du prédicteur du gène x_i est obtenu en faisant une cascade de « ou exclusif » de tous les comparateurs du gène x_i.

Code1 : représente le code VHDL synthétisable de l'entité *Gene_comp*.

```
ENTITY gene_comp is

 PORT(RND          : IN STD_LOGIC_VECTOR(7 downto 0);
      NB           : IN STD_LOGIC_VECTOR(7 downto 0);
      NSORTIE      : OUT STD_LOGIC;
      SORTIE       : OUT STD_LOGIC);
 END gene_comp;

ARCHITECTURE MAX2 of gene_comp is

begin
process(RND,NB)
begin

 if ( RND>NB ) then
    SORTIE <='1';
    NSORTIE <='0';
 else
    SORTIE <='0';
    NSORTIE <='1';
 end if;
 end process;

END MAX2;
```

4.4.1.2 L'entité Gene_f

Elle représente la fonction logique (ou prédicteur) pour un gène x_i, il y a autant d'entités *Gene_fi_j* (i : rang du gène, j : numéro de la fonction logique pour le gène i) que de predicteurs.

Entrées de l'entité :

- FX_k : variables d'entrées du predicteur

  k=1,...,n *avec* n = nombre de gènes
- EN : permet de sélectionner le predicteur (EN = 1)
- F : valeur de la fonction logique sous forme binaire

Une fois la fonction logique choisie (En = 1) et dépendamment des entrées x_k (k=1,...,n) de la fonction, un seul bit est sélectionné et représente l'état du gène x_i à l'instant t. Ainsi pour tous les gènes une architecture similaire de la figure 25 est réalisée. À chaque top horloge tous les états de tous les gènes sont déterminés et sont mis aux niveaux des sorties correspondantes.

Code2 : représente le code VHDL synthétisable de l'entité Gene_fi_j.

```
Entity gene_f3_2 is
 PORT(
          FX1 : in std_logic;
          FX2 : in std_logic;
          FX3 : in std_logic;
          EN : in std_logic;
          F : in std_logic_vector(7 downto 0);
          FCT : out std_logic);
END gene_f3_2 ;

ARCHITECTURE MAX2 of gene_f3_2 is
   COMPONENT LCELL
       PORT(a_in : IN STD_LOGIC;
          a_out : OUT STD_LOGIC);
   END COMPONENT;
SIGNAL fct1,sor1 : std_logic ;

Begin
U1:LCELL
 PORT MAP(fct1,sor1);
```

```
Process(FX1, FX2, FX3, EN,F)
        variable x_vec : std_logic_vector(2 downto 0);
Begin
        x_vec:= FX1 & FX2 & FX3 ;
if (EN = '1' ) then
 case x_vec is
          when "000" => fct1 <= F(0);
          when "001" => fct1 <= F(1);
          when "010" => fct1 <= F(2);
          when "011" => fct1 <= F(3);
          when "100" => fct1 <= F(4);
          when "101" => fct1 <= F(5);
          when "110" => fct1 <= F(6);
          when others => fct1 <= F(7);
 end case;
else
 fct1 <= '0';
end if;
end Process;
FCT <= sor1;
END MAX2;
```

Cette entité peut être différente d'un gène à un autre, cela dépend du nombre d'entrées FX_k pour chaque prédicteur. Il suffit de voir Le bloc « case » qui dépend du nombre d'entrée FX_k.

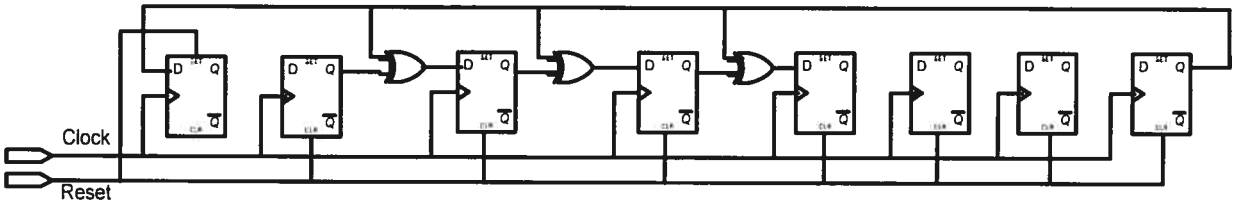
4.4.1.3 Générateur de nombres pseudo aléatoires

Pour que la simulation des réseaux de régulation génétique reflète la réalité des interactions génétique, tout système modélisant ces interactions doit impérativement prendre en considération la notion de non déterminisme qui est intrinsèque dans les systèmes de régulation génétique. Dans le cas des PBNs, il s'agit de choisir lequel des N réseaux qui sera considéré à un instant donné. En d'autres termes pour chaque gène, lequel des prédicteurs sera choisi à cet instant. Pour pouvoir introduire et réaliser cette notion d'incertitude dans notre système, le recours aux générateurs des nombres aléatoires devient indispensable. La meilleure façon de réaliser un générateur de nombre aléatoire serait d'utiliser les registres de décalage à gauche ou les LFSRs.

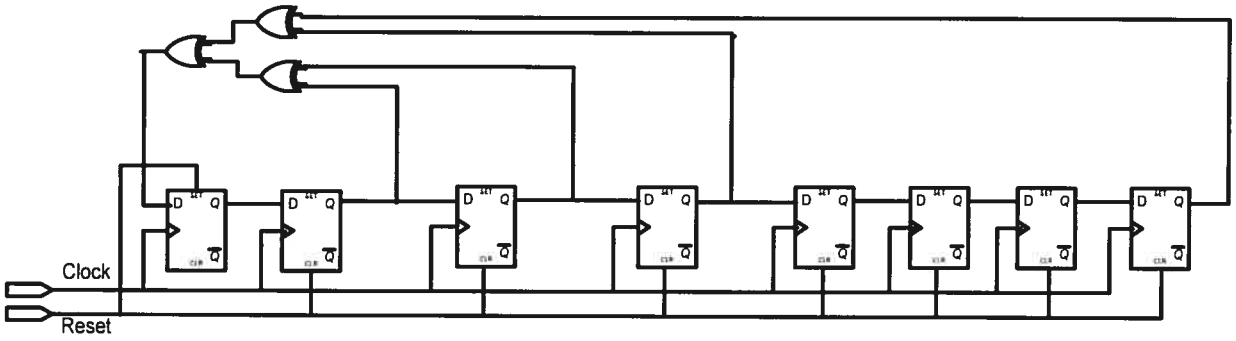
4.4.1.4 Registres à décalage à rétroaction linéaire (LFSR)

Un circuit LFSR est un registre à décalage séquentiel possédant une rétroaction combinatoire dont l'effet est de générer des séquences binaires pseudo aléatoires de différentes longueurs. La boucle de rétroaction force le registre à générer une suite pseudo aléatoire de valeurs binaires. Le choix des points de rétroaction détermine combien de valeurs d'une séquence donnée sont générées avant que cette séquence soit répétée. Certains choix de points de rétroaction produisent des séquences de longueur maximum, soit 2^n - 1 pour un registre de n bits.

La figure 26 illustre les deux structures possibles d'un LFSR de n bits avec n = 8.



(a) Un-à-Plusieurs



(b) Plusieurs-à-Un

Figure 26 : Structure d'un LFSR 8 bits : (a) Un-à-Plusieurs, (b) Plusieurs-à-Un.

Notre générateur de nombres aléatoires est composé de deux LFSRs (Un-à-Plusieurs). L'un de longueur 127 bits et l'autre de 128 bits. Le résultat est un autre générateur LFSR dont la période est égale au produit des périodes des deux LFSRs soit $2^{127} * 2^{128} = 2^{255}$ .Cette structure peut être théoriquement analysée comme étant un seul générateur LFSR [9].

À chaque instant notre générateur de nombres aléatoires génère 16 nombres aléatoires (de 8 bits chacun) qu'on associe aux différents gènes.

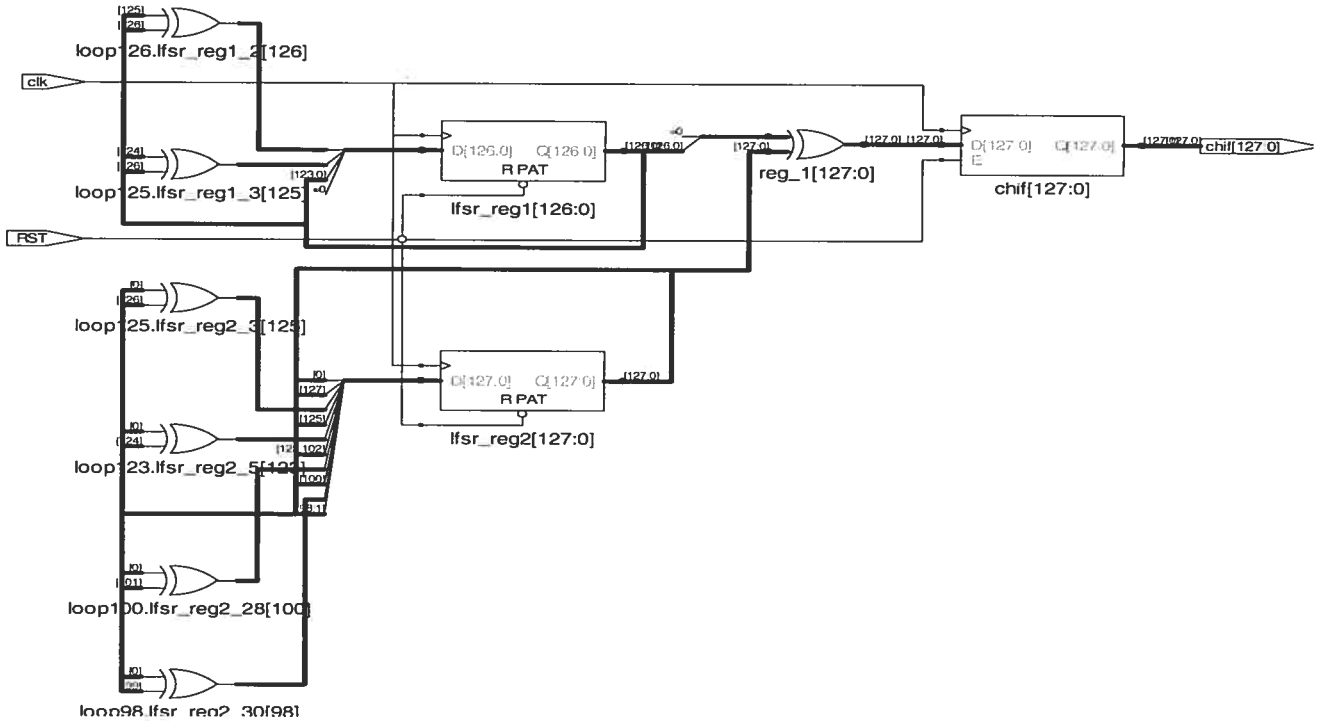


Figure 27 : Générateur des nombres aléatoires.

La figure 27 montre l'implémentation matérielle de notre générateur de nombres aléatoires qui est utilisé dans le système.

4.5 Simulation

L'algorithme de la partie simulation se déroule de la façon suivante :

Pour chaque gène *i* et selon le nombre aléatoire qui lui est associé

- Choisir le predicteur et ainsi la valeur du gène cible (suivant les variables d'entrées du predicteur)
- La valeur du gène x_j à l'instant t sera considérée comme entrée d'un prédicteur pour prédire le gène x_i à l'instant $(t+1)$.

Donc pour chaque gène, on applique cet algorithme. À chaque top horloge les états de tous les gènes sont déterminés et sont disponibles au niveau de la sortie *résultat i* correspondant au gène *i.* Les sorties ainsi formées représentent l'état du système à un instant donné.

4.6 Détection de cycles

Le but de cette procédure est de détecter les séquences de bits qui se répètent. Pour réaliser cela, l'algorithme de comparaison utilise des registres à décalage (figure 28). On associe à chaque gène un registre à décalage. Le nombre de mots (la profondeur du registre à décalage) est égale à la taille du cycle fixé par l'utilisateur avant la génération du code VHDL soit *tc*.
À chaque instant *t* l'entrée des registres à décalage est formée par la suite de bits produite par la simulation (soit le niveau 0) et à chaque top horloge le contenu des registres du niveau *i* est transmis au niveau *(i+1)*. Dépendamment de la profondeur des registres *(tc),* on compare la suite de bits générée lors de la simulation avec tous les autres mots formés à partir des registres à décalage (le mot du niveau p est formé par le contenu de tous les registres à décalage du niveau p).

Si on constate une égalité entre le mot du niveau 0 et celui du niveau p alors tous les mots se trouvant entre le niveau niv = 0 et niv = p seront pris et sont considérés comme un cycle.

L'algorithme de comparaison se déroule de la façon suivante :

Selon la taille du cycle attracteur *(tc)*

- S'il existe un cycle attracteur alors
  - Choisir d'une façon aléatoire un nouvel état du système et redémarrer la simulation (pour permettre de visiter un plus grand nombre de nœuds possible)
  - Permettre l'écriture du cycle attracteur dans la (es) FIFO (s).

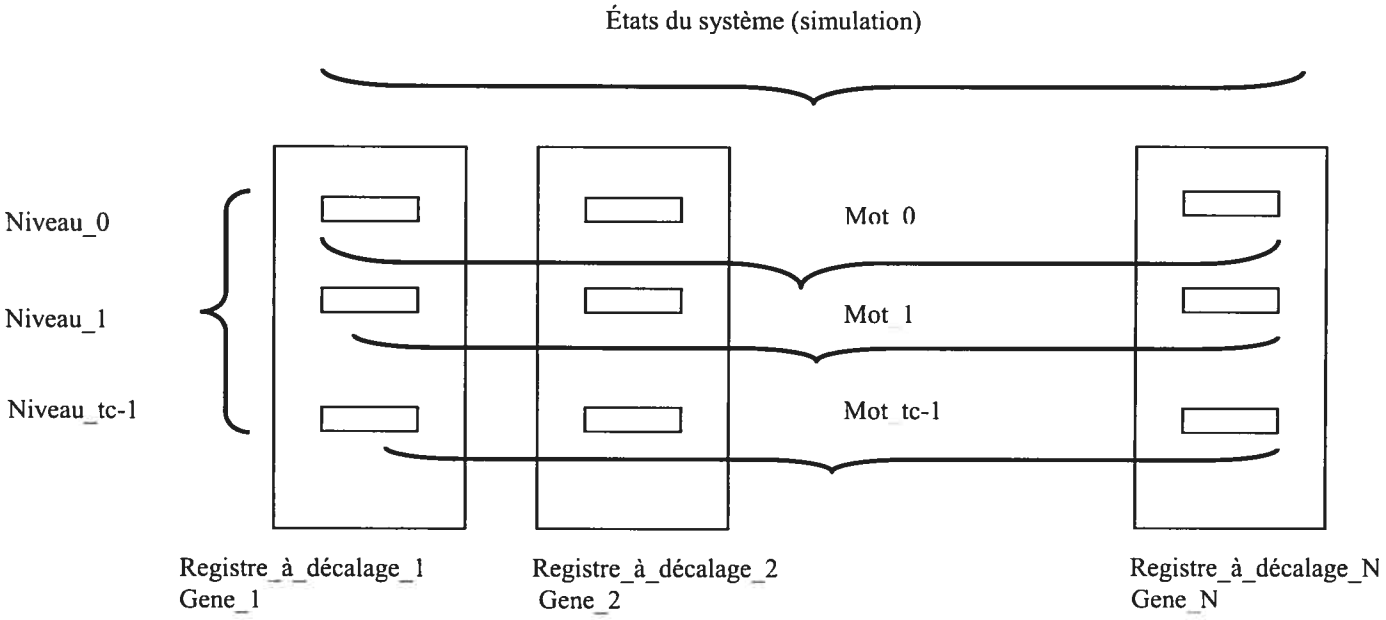


Figure 28 : Détection de cycle

Prenons un système formé de *N* gènes, donc le système registre-état a une largeur de *N* bits. Un cycle arrive si l'état du système se répète.

Par exemple, un cycle peut avoir les formes suivantes : S0, S450, S23,…,S28, S28, ...(point fixe) ou S52, …, S89, S57, S89,… (cycle dont la taille = 2), dans le deuxième cas on accepte que les cycles dont la taille est inférieure ou égale à *tc*.

À 25 Mhz, 25 millions d'états sont calculés chaque seconde et il serait impossible de vérifier et retenir tous les cycles possibles. Nous sommes seulement intéressés par la découverte de petits cycles qui révèlent les configurations stables du système. L'utilisateur peut instancier un mot mémoire (FIFO) de taille M qui garde la trace des derniers M états. À chaque cycle d'horloge, les M mots sont comparés d'une façon parallèle avec le nouvel état pour détecter si un cycle est arrivé. S'il en est ainsi, la simulation est bloquée pour M cycles c'est le temps exigé pour envoyer le cycle au processeur à travers la FIFO. Un nouvel état aléatoire de départ est alors calculé et la simulation continue.

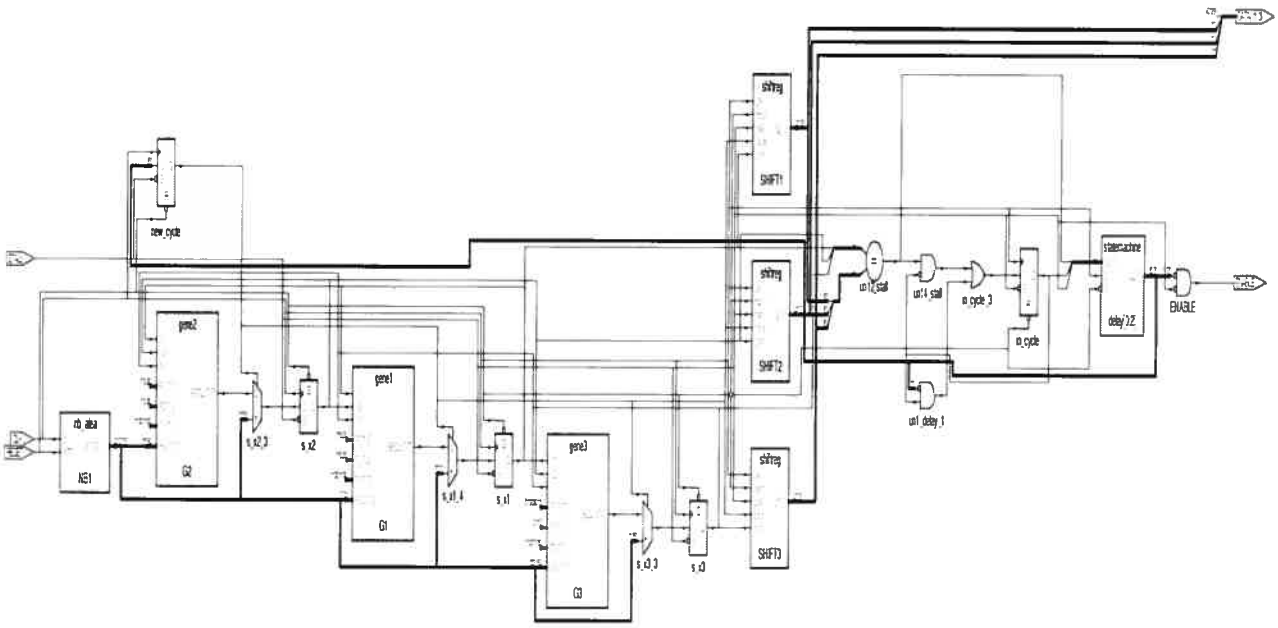


Figure 29 : Implémentation matérielle d'un système formé de trois gènes

4.7 L'interface FPGA – Processeur

La conception entière est connectée au bus AMBA du processeur ARM par les FIFOs. Ces FIFOs sont utiles de deux façons :

Premièrement, elles permettent au FPGA d'envoyer les états du cycle d'une façon très rapide (un état du système à chaque cycle d'horloge). Donc un temps minimal est perdu dans des transferts de données. Deuxièmement, le bus de données du processeur a seulement 32 bits de largeur, quand une simulation implique plus de 32 gènes plusieurs FIFO de 32 bits sont alors générées.

Le FPGA écrit dans toutes les FIFOs d'une façon parallèle mais le processeur leur accède d'une façon séquentielle.

4.7.1 Écriture des données

Notre système s'exécute sur une plateforme CMC, cette dernière nous offre des FIFOs de taille statique (16,32,…,256 bits), c'est-à-dire qu'on doit fixer leur taille dès le début de la génération du code VHDL.

Notre système modélise un réseau PBN de taille quelconque, alors le nombre de FIFO(s) utilisé (s) dépend du nombre de gènes. Nous avons utilisé une FIFO avec les caractéristiques suivantes :

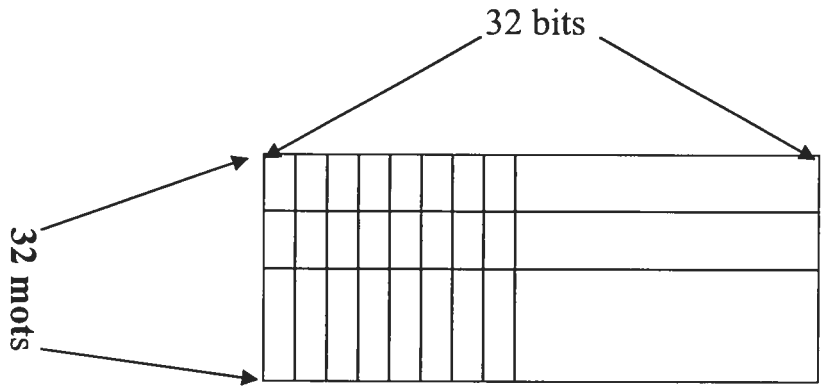


Figure 30: FIFO

Donc le nombre de FIFOs nécessaire pour l'écriture d'une suite de bits générée est égale à :

Nombre de gène / 32.

Écriture dans les FIFOs

Une fois le cycle détecté (comparaison), on permet l'écriture de cycle dans la (es) FIFO (s). Si au moins une FIFO devienne pleine, on bloque la première partie du système(STALL = 1) pour permettre à la troisième partie de lire et ainsi libérer la (es) FIFO(s) (STALL = 0) et par conséquent la première partie se débloque.

Supposons qu'on a $N = 100$ gènes, le nombre de FIFO nécessaire est égal :

Nombre de FIFO = 100 /32 = 3.12 donc 4 FIFOs.

L'écriture se fait de la manière suivante :

Mot_{Fifo1} = Suite_de_bits_générée(31 .. 0)

Mot_{Fifo2} = Suite_de_bits_générée(63.. 32)

Mot_{Fifo3} = Suite_de_bits_générée(95 .. 64)

Mot_{Fifo4} = Suite_de_bits_générée(127 ..96).

Éventuellement, des "zéros" ("0") sont ajoutés à gauche du Mot_{FIFO4} pour compléter le contenu de la 4^{ieme} FIFO.

Le code VHDL suivant permet de générer les FIFO nécessaires au fonctionnement du système. Ici, notre système est formé de 40 gènes, alors le nombre de FIFO nécessaire est égal à 2. L'état de notre système sera donc formé de 64 bits (2 X 32 bits). Comme on a que 40 gènes, alors 64-40 = 24 "zéros" seront ajoutés à gauche de la deuxième FIFO).

Code3 : représente le code VHDL synthétisable de l'entité Projet_Fifo

```
ENTITY Projet_Fifo is
        PORT( clk,reset,indre,status : in std_logic;
              addr : in std_logic;
              data_in : in std_logic_vector(63 downto 0);
              data_in_we : in std_logic;
              data_out : out std_logic_vector(31 downto 0);
              data_out_full : out std_logic;
              data_out_empty : out std_logic);
END Projet_Fifo;

ARCHITECTURE MAX2 of Projet_Fifo is
COMPONENT test_fifo
            PORT( CLK,sinit,wr_en,rd_en : in std_logic;
                  din                   : in std_logic_vector(31 downto 0);
                  dout                  : out std_logic_vector(31 downto 0);
                  full                  : out std_logic;
                  empty                 : out std_logic);
END COMPONENT ;
SIGNAL    s_out,s_out1,s_out2       : std_logic_vector(31 downto 0);
SIGNAL    sdata_out_empty           : std_logic;
SIGNAL    empty1,empty2,full1,full2 : std_logic;
SIGNAL    outrde                    : std_logic_vector(1 downto 0);
begin
Fifo1:test_fifo
      PORT MAP (
                clk => clk,
                sinit => reset,
                din => data_in(31 downto 0),
                wr_en => data_in_we,
                rd_en => outrde(0),
                empty => empty1,
                full => full1,
                dout => s_out1);

Fifo2:test_fifo
      PORT MAP (
                clk => clk,
                sinit => reset,
                din => data_in(63 downto 32),
                wr_en => data_in_we,
                rd_en => outrde(1),
                empty => empty2,
                full => full2,
                dout => s_out2);
```

FIFO 1

FIFO 2

```
PROCESS(addr,indre,s_out1,s_out2,empty1,empty2)

Begin

    if (status='1') then
        s_out <= "00000000000000000000000000000000"&empty2&empty1;
        sdata_out_empty<= '0';
        outrde <= "00";
    else
        case Addr is
            when '0' => s_out <= s_out1;
                        sdata_out_empty <= empty1;
                        if ( indre ='1' ) then
                            outrde <= "01";
                        else
                            outrde <= "00";
                        end if;

            when others => s_out <= s_out2;
                           sdata_out_empty <= empty2;
                           if ( indre ='1' ) then
                               outrde <= "10";
                           else
                               outrde <= "00";
                           end if;
        end case;
    end if;

END PROCESS;

data_out_empty <= sdata_out_empty;
data_out_full <= full1 or full2;
data_out <= s_out;

END MAX2;
```

Écrire l'état des 2 FIFOs

Écriture dans la FIFO 1

Écriture dans la FIFO 2

4.8 Partie logicielle

La partie logicielle suit le diagramme donné dans la figure 31. Les spécifications d'entrées, sont un fichier contenant la topologie du réseau. Un programme C s'exécute une un PC conventionnel génère le code VHDL synthétisable de tout le système. Ce code est compilé par des outils Xilinx et finalement chargé dans le FPGA. Un programme C pour le processeur ARM est compilé est transféré dans la RPP. Ce programme devra prendre les cycles découverts par le FPGA.

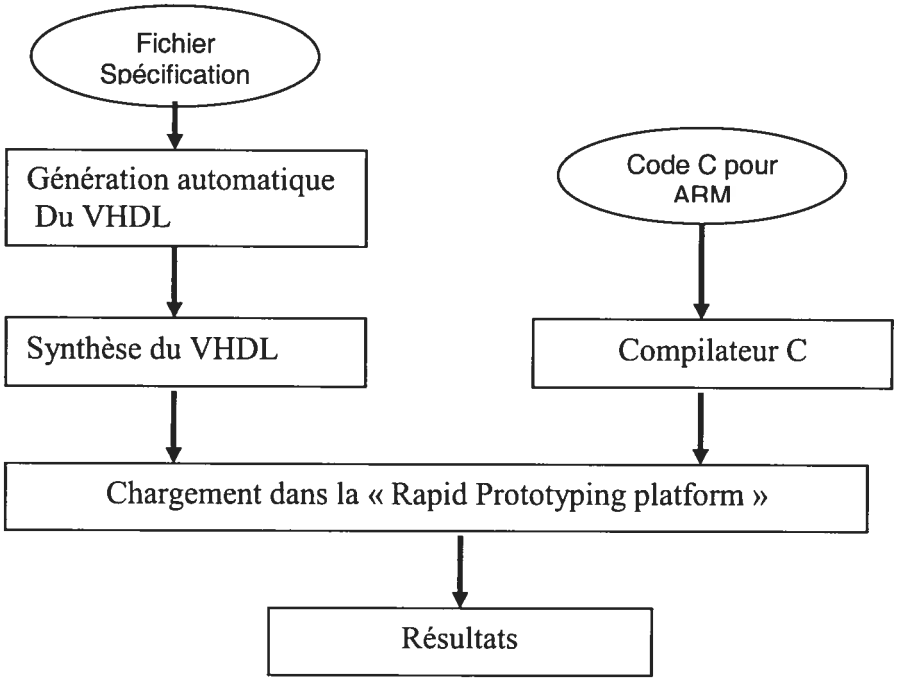


Figure 31 : Planification des tâches

4.9 Génération automatique du code VHDL

Le fichier de spécification est lu par un programme C. Celui-ci produira trois sortes de fichiers VHDL: l'interface au processeur (avec une ou plusieurs FIFOs); L'interconnexion de tous les gènes avec leurs voisins et enfin le code VHDL de chaque gène.

L'utilisateur doit spécifier la longueur maximale des cycles que le FPGA peut détecter (le paramètre M vu dans la section précédente).

Le générateur du code VHDL accepte en :

Entrées :

- N : le nombre de gènes
- File : nom de fichier de données qui contient par gène
  - Nf : le nombre de fonctions logiques (predicteurs)
  - Nv : le nombre de variable d'entrées de chaque fonction logique
  - Les variables d'entrées de chaque fonction logique
  - c_j^i : la (es) probabilité (s) de sélection qu'une fonction logique soit choisie
  - La valeur des prédicteurs sous forme binaire
- tc : la taille du cycle (le point fixe correspond à une taille égale à 1)

Sortie : Tous les fichiers VHDL nécessaires à l'exécution de notre système hardware

4.10 Acquisition de données et traitement

Le processeur peut avoir accès aux bits du statut de la FIFO. Aussitôt qu'un nouveau cycle est prêt dans la FIFO, le processeur lit d'une façon séquentielle tout le cycle (nombre de lecture dépend du nombre de FIFOs). Le cycle est alors comparé aux cycles précédemment sauvegardés en utilisant une table de hachage. Si le cycle existe déjà, son compteur est alors incrémenté. Autrement le cycle est sauvegardé dans la base de données des cycles et son compteur est mis à 1. À tout moment l'utilisateur peut télécharger les résultats : le cycle ainsi que son nombre d'occurrence pendant le temps de simulation.

4.11 Lecture des données

Pour décrire la lecture des données, utilisant un exemple où le nombre de gènes est égal : $N = 100$, alors le nombre de FIFOs utilisé est de 4, donc la taille de la donnée est de 128 bits (figure 32). Pour pouvoir lire toute la donnée correctement à partir des FIFOs, on doit faire 4 lectures ordonnées ($FIFO_4$, $FIFO_3$, $FIFO_2$, $FIFO_1$), la lecture se fait en envoyant l'adresse de la FIFO. Le programme C convertit et concatène le contenu des 4 FIFOs dans le bon ordre, puis il détecte les différents cycles.

Il détermine leur taille ainsi que leurs occurrence dans le cas ou il y a une répétition d'un cycle et enfin il sauvegarde les cycles non dupliqués, leur taille ainsi que leur occurrence dans un fichier (voir annexe C).

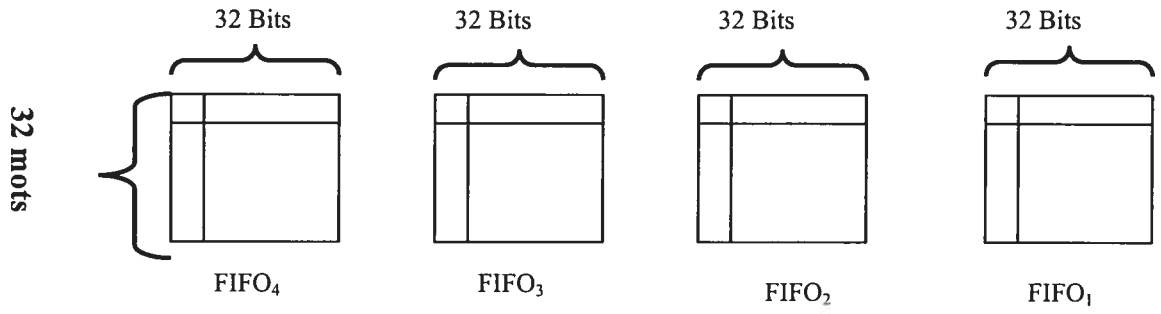


Figure 32 : Lecture des données

Le programme C qui permet d'effectuer l'opération de lecture est donné ci-dessous (ici on a un exemple de 40 gènes, donc 2 lectures puisque on a 2 FIFOs):

Code4 : représente le code C qui s'exécute sur l'ARM

```
int main(void)
{ int j,addr11,nr_g; int addr1=0xc2100000; int addr2=0xc2100080;
 long tmp; long status=0; int i=0; nr_g=2;
 while (i==0){
        index=0;
        for (j=nr_g-1;j>=0;j--){
              addr11=addr1+(j<<2);
        while ((status=word_read(addr2))&0x00003);
              tmp=word_read(addr11);
        convdecbin(tmp);conc();
        index=index+32;
        }; printf("\n le mot lu est = %s",temp);
}}
```

On s'assure que les 2 FIFOs ne sont pas vides)

Partie de détection et sauvegarde des cycles une fois lus (Annexe C).

Résultats & performance

5.1 Résultats expérimentaux

Trois types de comparaisons en terme de temps d'exécution ont été effectués. Une première comparaison a été effectuée entre notre approche et celle utilisant des fonctions Matlab données en [16]. La deuxième comparaison a porté uniquement sur notre approche mais entre une version purement logicielle (codée en C) et celle utilisant un FPGA. Finalement, une troisième comparaison a été faite entre la version purement logicielle de notre approche et celle donnée en [16].

Le FPGA qu'on utilise dans notre système est un Virtex 2000 (Xilinx) d'une fréquence de 25 Mhz. La table *XI* montre les résultats d'une comparaison faite sur l'exemple vu dans le chapitre précédent, entre notre approche et celle utilisant des outils Matlab donnés en [16]. Les résultats montrent que notre approche réalise un million d'étapes en 40 ms contre 514 secondes pour l'autre approche s'exécutant sur un Pentium 1.5 Ghz. Un gain de quatre ordres de magnitude a été obtenu.

Table *XI*: Comparaison entre version FPGA et version Matlab

Nombre d'étapes					
10 000		100 000		1 000 000	
Matlab	FPGA	Matlab	FPGA	Matlab	FPGA
6 s	0.4 ms	51 s	04 ms	514 s	40 ms

La table *XII* montre les résultats de la comparaison faite entre les deux variantes de notre approche. Il faut noter que la version purement logicielle de notre approche s'exécute sur un Pentium 1.5 Ghz. On remarque que pour un million d'étapes, le temps d'exécution dans la version logicielle augmente d'une façon linéaire comparativement au nombre de gènes

considérés dans le système et que le temps d'exécution reste constant dans la version matérielle (on utilise des équations logiques simples). Le gain obtenu est de quatre ordres de grandeur.

Table XII: Comparaison entre version logicielle et version FPGA

Version logicielle		Version FPGA
Nb de gène	Nb étapes = 1 000 000	Nb étapes = 1 000 000
10	5 s	40 ms
100	50 s	40 ms
1000	500 s	40 ms

Une troisième comparaison a été réalisée entre la version purement logicielle de notre approche et celle utilisant des fonctions Matlab données en [16]. La table *XIII*, montre les résultats obtenus.

Table XIII: Comparaison entre version logicielle et version Matlab

Nombre d'étapes					
10 000		100 000		1 000 000	
Matlab	logicielle	Matlab	logicielle	Matlab	logicielle
6 s	0.02 s	51 s	0,2 s	514 s	2 s

De plus, l'algorithme utilisé dans les fonctions Matlab [16] construit une matrice de 2^{2N} éléments qui limite N (nombre de gênes) à 15. Pour 15 gènes l'ordre de magnitude de la taille mémoire est le milliard d'octets, alors que dans notre approche, nous ne construisons pas explicitement le graphe d'états et par conséquent, nous ne rencontrerons pas ce genre de problèmes.

Notre FPGA (V2000efg680-6) qui est de taille moyenne, permet d'implémenter des réseaux de 800 gènes avec deux prédicteurs par gène et 5 variables par prédicteur.

La Table *XIV* montre l'espace utilisé dans le FPGA en terme de composants tels que des LUT à 4 entrées. Il donne aussi la fréquence maximum d'horloge. Une étape de tout le réseau est exécutée en un seul cycle d'horloge. Cependant, il faut noter que pour 1000 gènes, on excède la capacité du FPGA en termes de cellules logiques et aussi en LUTs.

Table XIV : Occupation du FPGA

N		Nv = 3	%	Nv = 4	%	Nv = 5	%
10	S	408	2	413	2	425	2
	F	453	1	453	1	463	1
	L	527	1	529	1	572	1
	I	66	12	66	12	66	12
	M	89		89		89	
100	S	2291	11	2341	12	2513	13
	F	1696	4	1703	4	1735	4
	L	3888	10	3999	10	4338	11
	I	68	13	68	13	68	13
	M	68		68		68	
500	S	10599	55	10819	56	11782	61
	F	6686	17	6701	17	6747	17
	L	17839	46	18344	47	19984	52
	I	70	13	70	13	70	13
	M	39		39		39	
800	S	16879	87	17236	89	18835	98
	F	10429	27	10455	27	10536	27
	L	28408	73	29213	76	31980	83
	I	71	13	71	13	71	13
	M	29		29		29	
1000	S	21076	10	21514	11	23497	122
	F	12975	9	13007	2	13103	34
	L	35442	33	36443	33	39871	103
	I	71	92	71	94	71	13
			13		13		
	M	25		25		25	

Dans la table XIV, on suppose qu'il y a *2* prédicteurs par gène, *N* indique le nombre de gène dans le système, *Nv* représente le nombre de gène par prédicteur, *S* représente les cellules logiques utilisées, *L* correspond au nombre de « look-up tables » à 4 entrées utilisées, *F* correspond au nombre de registres utilisées (flip-flop), *I* représente le nombre d'entrées et sortie et enfin *M* représente la fréquence maximale atteinte en Mhz.

Il est important de signaler que le code s'exécutant sur le processeur ARM alourdit notre système. En effet, la partie simulation et détection des cycles (figure 23) se bloque à chaque fois que les mémoires FIFOs se remplissent (STALL=1). Son déblocage est subordonné par le

traitement de ces cycles au niveau du processeur ARM (lecture, comparaison des cycles non dupliqués, occurrence des cycles). Le temps d'inoccupation de la partie simulation et détection (coté FPGA) est très important. Il représente 5250 fois leur temps de traitement (résultat obtenu d'un système formé de 300 gènes).
Dans le but d'améliorer notre système, une solution serait de réaliser un sous-système matériel dans le même FPGA capable de sauvegarder les cycles non dupliqués dans une FIFO et d'en déterminer leurs occurrences.

5.2 Performances

Notre approche (ses deux variantes) a permis d'éliminer le problème de dimension que connaissaient les solutions logicielles dans la modélisation des interactions génétiques.
Nous avons réalisé un système matériel générique capable de modéliser un PBN de taille quelconque. Le nombre de prédicteurs par gène peut être différent d'un gène à un autre. Aussi le nombre de variables d'entrées des fonctions logiques peut varier pour le même gène ou pour un gène différent et cela comparativement aux fonctions Matlab citées en [16]. Les variables d'entrées des prédicteurs peuvent être différentes pour le même gène ou pour un gène différent. Puisque la taille maximale du cycle attracteur est fixée au début par l'utilisateur ce qui permettrait à l'utilisateur d'obtenir tous les cycles dont la taille est inférieure ou égale à la taille donnée. Cette façon de faire donne plus de flexibilité dans la recherche des attracteurs.
Le redémarrage aléatoire représente un des points importants dans notre système. Il permet de visiter un plus grand nombre de nœuds possibles ce qui nous permet d'obtenir un plus grand nombre de cycle.

En plus de la souplesse et le confort qu'offre tout système générique, notre système peut s'exécuter avec une fréquence qui avoisine les 100 Mhz. Notre générateur de nombres aléatoires contribue d'une façon favorable à créer un environnement non déterministe qui correspond exactement à l'environnement génétique. L'approche de co-design utilisé et la qualité de généricité rendent encore plus facile l'utilisation de notre système. En effet, l'utilisateur de notre système peut être de différentes disciplines, il peut être biologiste, mathématicien ou informaticien.

Conclusion et perspectives

6.1 Conclusion

Dans ce mémoire, différentes approches de modélisation des interactions génétiques ont été discutées. On a montré la façon avec la quelle chacun de ces modèles représente ces interactions. On a aussi donné les avantages et les inconvénients de chacun d'eux.

Nous avons présenté une nouvelle classe de modèles pour les réseaux de régulation génétique appelés les réseaux booléens probabilistes (PBNs) qui sont des généralisations probabilistes des réseaux booléens standards. L'étude de tous ces modèles montre l'importance de la modélisation dans la compréhension des interactions génétiques dans les réseaux de régulation.

Cependant, tous ces modèles rencontrent le « problème de dimension ». En principe, un réseau de régulation génétique doit contenir tous les gènes disponibles dans les données temporelles utilisées pour l'inférence de la structure du réseau. Mais à cause des ressources statiques dont on dispose, la modélisation des réseaux de régulation génétique se limiterait à de petits réseaux.

Dans ce mémoire, nous avons montré les limites des solutions logicielles présentées récemment dans la littérature dans la modélisation statique des réseaux de régulation génétique. Nous avons présenté une solution basée sur l'approche de conception conjointe matériel-logiciel capable de répondre aux attentes des chercheurs. À travers cette solution, nous avons montré la capacité des FPGAs à manipuler des réseaux de régulation génétiques gigantesques pour découvrir des propriétés telles que les attracteurs qui jouent un rôle très important dans la génomique. On a aussi montré la puissance de calcul parallèle que peut offrir un FPGA.

L'approche que nous avons présentée a permis de résoudre le problème de réservation mémoire qu'on rencontre souvent dans les solutions logicielles et diminuer d'une façon considérable le temps d'exécution.

Les propriétés de re-configurabilité des FPGAs et la généricité de notre système matériel ont permis de concevoir et d'implémenter des systèmes modélisant des réseaux de régulation génétiques de grande taille, en un temps très court.
L'approche de co-design matériel-logiciel utilisé nous a permis d'obtenir un gain de quatre ordres de grandeur comparativement à la solution purement logicielle.

6.2 Perspectives

L'inférence de la structure des réseaux de régulation génétique à partir des données temporelles est une tâche qui exige beaucoup de ressources logicielles. Il serait intéressant de compléter notre système en lui intégrant un sous système matériel, capable d'inférer la structure du réseau de régulation génétique à partir des données temporelles d'expression de gènes et d'en déduire les différentes interactions entre les différents gènes utilisant une des méthodes d'inférences vues dans ce mémoire.

Référence

[1] I.Shmulevich, ER.Dougherty, and W. Zhang, "From Boolean to probabilistic Boolean networks as models of genetic regulatory networks", Proceedings of the IEEE, Vol. 90, No. 11, pp. 1778-1792, 2002.

[2] I. Shmulevich, ER.Dougherty, S. Kim, W. Zhang, "Probabilistic Boolean Networks: A Rule-based Uncertainty Model for Gene Regulatory Networks", Bioinformatics, Vol. 18, No. 2, pp. 261-274, 2002.

[3] I. Shmulevich, E. R. Dougherty, W. Zhang, "Gene Perturbation and Intervention in Probabilistic Boolean Networks", Bioinformatics, Vol. 18, No. 10, pp. 1319-1331, 2002

[4] S. Huang, "Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery", Journal of Molecular Medicine, vol. 77, pp. 469-480, 1999.

[5] S. Huang, "Cell state dynamics and tumorigenesis in Boolean regulatory networks", InterJournal Genetics, MS: 416. Manuscript for the proceedings of ICCS 2000

[6] S. Liang, S. Fuhrman, and R. Somogyi, "REVEAL, A General Reverse Engineering Algorithm for Inference of Genetic Network Architectures", Pacific Symposium on Biocomputing 3, pp. 18-29, 1998.

[7] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano, "Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions", Proc. the 9th Annual ACM- SIAM Symposium on Discrete Algorithms (SODA'98), pp. 695-702, 1998.

[8] T. Akutsu, S. Miyano, and S. Kuhara, "Identification of Genetic Networks from a Small Number of Gene Expression Patterns Under the Boolean Network Model", Pacific Symposium on Biocomputing 4, pp. 17-28, 1999.

[9] P. L'Ecuyer, "Uniform Random Number Generators", Proceedings of the 1998 Winter Simulation Conference, IEEE Press, pp. 97-104, 1998.

[10] CMC Rapid-Prototyping Platform: Design Flow Guide, Version 1.0, ICI-110. Canadian Microelectronics corporation, 2002

[11] Integrator /AP ASIC development motherboard. User Guide, ARM DUI 0098B, 1999-2001

[12] Integrator /CM7TMDI user Guide, ARM DUI 0126B, 1999-2001.

[13] S.A Kauffman, "The Origins of Order: Self-Organization and Selection in Evolution", New York: Oxford Univ. Press, 1993

[14] El Mostapha Aboulhamid, cours Ift6221, DIRO, Université de Montréal. http://www.iro.umontreal.ca/~aboulham/ift6221A03.htm

[15] JP. David, cours Ift6223, DIRO, Université de Montréal. http://www.iro.umontreal.ca/~pift6223/

[16] BN/PBN MATLAB Toolbox, follow link on http://www2.mdanderson.org/app/ilya/PBN/PBN.htm

[17] HIDDE DE JONG, "Modelling and simulation of genetic regulatory systems: A literature review", Journal of Computational Biology, volume 9, number 1, pp. 67-103, 2002. http://cll.stanford.edu/causal/dejong.jcb.pdf

[18] J.Stark, D. Brewert, M. Barencot, D. Tomescut, R.Callard and M. Hubank, "Reconstructing gene networks: what are the limits ?", Department of Mathematics, Imperial College London, pp. 1519-1525, 2003.

[19] K. Murphy. and S. Mian, "Modelling gene expression data using dynamic Bayesian networks", Technical Report. Berkely, 1999.

[20] N. Friedman, M. Linial, I. Nachman, and D. Peer, "Using Bayesian Networks to analyze Expression Data", Journal of Computational Biology, vol. 7, pp. 601-620, 2000.

[21] C. D. Johnson,Y. Balagurunathan, K. P. Lu, M. Tadesse, M. H. Falahatpisheh, R. J. Carroll, E. R. Dougherty, C. A. Afshari, and K. S. Ramos, "Genomic profiles and predictive biological networks in oxidant-induced atherogenesis", *Physiological Genomics* 13: pp. 263-275, 2003.

[22] Seungchan Kim, Edward R. Dougherty, Michael L. Bittner, Yidong Chen, Krishnamoorthy Sivakumar, Paul Meltzer, Jeffrey M. Trent, "General nonlinear framework for the analysis of gene interaction via multivariate expression arrays", Journal of Biomedical Optics 5(4), pp. 411-424, 2000.

[23] Edward R. Dougherty, Michael L. Bittner, Yidong Chen, Seungchan Kim, Krishnamoorthy Sivakumar, Junior Barrera, Paul Meltzer, Jeffrey M. Trent, "Nonlinear Filters in Genomic Control", Proceedings of the IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, Antalya, Turkey, June, 1999

[24] I. Shmulevich, A. Saarinen, O. Yli-Harja & J. Astola, "Inference of genetic regulatory networks via best-fit extensions", In: Shang, W. & Shmulevich, I. (Eds). Computational and Statistical Approaches to Genomics, Chapter 11, pp. 197-210, 2002

[25] J. Lacroix, "Les chaînes de Markov", cours DEA, Université Pierre et Marie Curie, Paris, France. http://www.proba.jussieu.fr/cours/dea/telehtml/telehtml.html

[26] H. de JONG and M. Page, "GNA (Genetic Network Analyzer): A computer tool for the modeling and simulation of genetic regulatory networks", Institut National de Recherche en Informatique et en Automatique (INRIA) Rhône-Alpes http://bacillus.inrialpes.fr/gna/html/building-model.htm

[27] " Architecture synchronisée par les données pour système reconfigurable", thèse de doctorat de Jean Pierre David, Faculté des Sciences Appliquées, Laboratoire de Microélectronique, Université Catholique de Louvain, Juin 2002.

[28] "Lexique encycloBio", http://www.cite-sciences.fr/francais/ala_cite/expo/tempo /defis/lexique

[29] Kanehisa, M., and Goto, S. "KEGG: Kyoto encyclopedia of genes and genomes". Nucl. Acids Res. 28(1), pp. 27–30, 2000.

[30] Kolpakov, F.A., Ananko, E.A., Kolesov, G.B., and Kolchanov, N.A. GeneNet: "A gene network database and its automated visualization". Bioinformatics 14(6), pp. 529–537, 1999.

ANNEXE A

Le code C du générateur VHDL

```
**************************************************************************
Genration Projet.vhd

**************************************************************************

int main()
{
 const int max_lignes= 10000;
 char *texte[max_lignes];
 char *cij[max_lignes];
 int indice2=1;
 int indice3=1;
 char vv[10000]="\0";
 char vvv[10000]="\0";
 int qt;
 printf("Entrez le Nombre de genes :");
 scanf("%d",&n);
 printf("\n");
 lignes=0;
 lig_cij=0;
 index=1;
 char NOM_FICHIER[30]; /* nom du fichier */
 do
 {
       printf("Entrez le nom du fichier : ");
       scanf("%s", NOM_FICHIER);
       in3 = fopen(NOM_FICHIER, "r");
       if (!in3)
          printf("\aERREUR: Impossible d'ouvrir le fichier: %s.\n", NOM_FICHIER);
}
       while (!in3);
              printf("Entrez le Nombre de Sequence :");
              scanf("%d",&nc);
              printf("\n");
              in=fopen ("Projet.vhd","w");
              indice =1;
for (i = 1 ; i<n+1; i++)
{
        _strset(para,' ');
         fscanf(in3,"%s", para);
         test=strlen(para);
         char vtemp [10000]="";
         m=0;k=0;
         while (para[m] != ','){
                       vtemp[k]=para[m];
                       m++;k++;};
                       m++;res=0;
                       nf[i]=atoi(vtemp);
                       for (j=1;j<nf[i]+1;j++){
                            k=0; char vtemp6 [10000]="";


                             while (para[m] != ','){
```

```
            vtemp6[k]=para[m];
            m++;k++;};
            m++;
            nv[i][j]=atoi(vtemp6);
            res=res+atoi(vtemp6);
        };
        test=test-m;nvg[i]=0;
        for (j=1;j<res+1;j++){
            char vtemp7 [10000]=""; k=0;
            while (para[m] != ','){
                    vtemp7[k]=para[m];
                    test--;
                    if (test == 0)
                        break;
                        m++;k++;
            };
            test--;
            nr_var[indice]=atoi(vtemp7);
            verif(i,nr_var[indice],indice3,indice2);
            indice++;indice3++;m++;
       };
        tri(indice2,(indice2+nvg[i]-1));
        _strset(coef,' ');
        fscanf(in3,"%s", coef);
        lig_cij= lecturec(cij, max_lignes,lig_cij,nf[i]);
        _strset(pred,' ');
        fscanf(in3,"%s", pred);
        lignes= lecture(texte,i,lignes,nf[i]);
        indice2=indice2+nvg[i];
        indice3=indice2;
};
fprintf(in,"Library IEEE;\n");
fprintf(in,"use IEEE.std_logic_1164.all;\n");
fprintf(in,"use IEEE.std_logic_arith.all;\n");
fprintf(in,"use IEEE.std_logic_unsigned.all;\n\n");
fprintf(in,"ENTITY Projet is\n\n");
fprintf(in," PORT( CLK : in std_logic;\n");
fprintf(in," RESET : in std_logic;\n");
fprintf(in," STALL : in std_logic;\n");
if (n<32)
   qt=1;
else
{ qt = n/32;
   if (n%32 !=0)
       qt++;
};
fprintf(in," DATA : out std_logic_vector(%d downto 0);\n",(qt*32-1));
fprintf(in," ENABLE : out std_logic);\n");
fprintf(in," END Projet;\n\n");
fprintf(in,"ARCHITECTURE MAX2 of Projet is\n");
fprintf(in,"\n");
impressionf(texte, lignes);
fprintf(in,"\n");
impressionc(cij, lig_cij);
```

```
fprintf(in,"\n");
 indice=0;
for (m=1; m<n+1;m++)
{
      fprintf(in,"COMPONENT gene%d \n\n",m);
      fprintf(in," PORT(\n");
      for (j=1; j<nvg[m]+1;j++)
           fprintf(in," X%d : in std_logic;\n",ent[indice+j]);
      for (k=1; k<=nf[m];k++){
           l=pow(2,nv[m][k])-1;
           fprintf(in," F%d : in std_logic_vector (%g downto 0) ;\n",k,l);
      };
           for (k=1; k<nf[m];k++)
                fprintf(in," C%d : in std_logic_vector (7 downto 0) ;\n",k);
           if (nf[m]==1)
               fprintf(in," C1 : in std_logic_vector (7 downto 0) ;\n");
           fprintf(in," RND : in std_logic_vector (7 downto 0);\n");
           fprintf(in," RESULTAT : out std_logic);\n");
           fprintf(in," END COMPONENT ;\n\n");
           indice=indice+nvg[m];
};
 fprintf(in," COMPONENT shiftreg \n");
 fprintf(in," PORT( clk : in std_logic;\n");
 fprintf(in," reset : in std_logic;\n");
 fprintf(in," stall : in std_logic;\n");
 fprintf(in," clear : in std_logic;\n");
 fprintf(in," shift : in std_logic;\n");
 fprintf(in," q : out std_logic_vector(%d downto 0));\n",nc-1);
 fprintf(in," END COMPONENT ;\n\n");
 fprintf(in," COMPONENT nb_alea \n");
 fprintf(in," PORT( CLK : in std_logic;\n");
 fprintf(in,"           RST : in std_logic;\n");
fprintf(in,"           CHIF : out std_logic_vector(127 downto 0));\n");
fprintf(in," END COMPONENT ;\n\n");
fprintf(in," SIGNAL S_RND : std_logic_vector(127 downto 0);\n");
for (j=1; j<n+1;j++)
     fprintf(in," SIGNAL S_X%d : std_logic;\n",j);
for (j=1; j<n+1;j++)
     fprintf(in," SIGNAL S_R%d : std_logic;\n",j);
for (j=1; j<n+1;j++)
     fprintf(in," SIGNAL S_Q%d : std_logic_vector(%d downto 0);\n",j,nc-1);
fprintf(in," SIGNAL DELAY : std_logic_vector(%d downto 0);\n",nc-1);
fprintf(in," SIGNAL");
for (j=0; j<nc-1 ;j++)
    fprintf(in," R_vec%d,",j);
fprintf(in," R_vec%d :std_logic_vector(%d downto 0);\n",j,n-1);
fprintf(in," SIGNAL NEW_CYCLE : std_logic;\n");
fprintf(in," SIGNAL IN_CYCLE : std_logic;\n\n");
fprintf(in,"begin\n\n");
fprintf(in,"NB1:nb_alea\n");
fprintf(in," PORT MAP (\n");
fprintf(in,"                CLK => CLK,\n");
fprintf(in,"                RST => RESET,\n");
fprintf(in,"                 CHIF => S_RND);\n\n");
```

```
                indice=0;

for (m=1; m<n+1;m++){
        fprintf(in,"G%d:gene%d \n\n",m,m);
        fprintf(in," PORT MAP (\n");
        for (j=1; j<nvg[m]+1;j++)
        fprintf(in," X%d => S_X%d,\n",ent[indice+j],ent[indice+j]);
        for (k=1; k<=nf[m];k++)
        fprintf(in," F%d => F%d%d,\n",k,k,m);
        for (k=1; k<nf[m];k++)
        fprintf(in," C%d => C%d%d,\n",k,k,m);
                if (nf[m]==1)
                   fprintf(in," C1 => C1%d,\n",m);
        fprintf(in," RND => S_RND(%d downto %d),\n",cptf,cpti);
        fprintf(in," RESULTAT => S_R%d);\n",m);
        cpti=cptf+1;
        if (cpti == 128)
          { cptf=7;cpti=0;}
        else
          cptf=cptf+8;
          indice=indice+nvg[m];
        };
        for (m=1; m<n+1;m++){
                fprintf(in,"\n");
               fprintf(in,"SHIFT%d:shiftreg\n\n",m);
               fprintf(in," PORT MAP (\n");
               fprintf(in," clk => CLK,\n");
               fprintf(in," reset => RESET,\n");
               fprintf(in," stall => STALL,\n");
               fprintf(in," clear => NEW_CYCLE,\n");
               fprintf(in," shift => S_X%d,\n",m);
               fprintf(in," q => S_Q%d);\n",m);
               fprintf(in,"\n");
               fprintf(in,"PROCESS(CLK,RESET)\n\n");
               fprintf(in,"begin\n\n");
               fprintf(in,"if (RESET='0') then\n");
               for (j=1; j<n+1;j++)
               fprintf(in," S_X%d <='1';\n",j);
               fprintf(in," elsif (CLK'event and CLK='1') then\n");
                fprintf(in," if (STALL = '0') then \n");
               fprintf(in," if (NEW_CYCLE = '1') then \n");
               for (j=1; j<n+1;j++){
                fprintf(in," S_X%d <= S_RND(%d);\n",j,ch_rnd);
                if (ch_rnd == 0)
                        ch_rnd=127;
                else
                        ch_rnd--;
               };
               fprintf(in," else \n");
               for (j=1; j<n+1;j++)
                fprintf(in," S_X%d <= S_R%d;\n",j,j);
                fprintf(in," end if; \n");
                fprintf(in," end if; \n");
                fprintf(in," end if; \n");
```

```
                fprintf(in,"END PROCESS;\n\n");
                fprintf(in," R_vec0<=");
               for (j=1; j<n;j++)
                fprintf(in," S_X%d &",j);
                fprintf(in," S_X%d; \n",j);
               for (k=1; k<nc;k++){
                fprintf(in," R_vec%d<=",k);
                for (j=1; j<n;j++)
                fprintf(in," S_Q%d(%d) &",j,k-1);
                fprintf(in," S_Q%d(%d);\n",j,k-1);
};
                fprintf(in,"\n");
               fprintf(in,"\n");
               fprintf(in,"PROCESS(CLK,RESET)\n\n");
               fprintf(in,"begin\n\n");
               fprintf(in,"if (RESET='0') then\n");
               fprintf(in," DELAY <= (others=>'0');\n");
               fprintf(in," IN_CYCLE <='0';\n");
               fprintf(in," NEW_CYCLE <='0';\n\n");
               fprintf(in,"elsif (CLK'event and CLK='1') then\n\n");
               fprintf(in," if (STALL = '0') then \n");
               fprintf(in," if (IN_CYCLE='1') then\n");
                vv[0]='1';
               for (j=1; j<nc;j++)
                vv[j]='0';
                fprintf(in," if (DELAY=\"%s\") then\n",vv);
                fprintf(in," IN_CYCLE <='0';\n");
                fprintf(in," DELAY <=(others=>'0');\n");
                fprintf(in," else \n");
                for (j=nc-1; j>0;j--)
                fprintf(in," DELAY(%d)<= DELAY(%d);\n",j,j-1);
                fprintf(in," DELAY(0)<= '0';\n",j,j-1);
                fprintf(in," end if; \n");

                for (k=1; k<nc;k++){
                fprintf(in," elsif (");
                fprintf(in," R_vec0 = R_Vec%d) then\n",k);
                for (j=0; j<=k;j++)
                fprintf(in," DELAY(%d)<='1';\n",j,j-1);
                fprintf(in," IN_CYCLE<='1';\n");
};
                fprintf(in," else \n");
                for (j=nc-1; j>0;j--)
                fprintf(in," DELAY(%d)<= DELAY(%d);\n",j,j-1);
                fprintf(in," DELAY(0)<= '0';\n",j,j-1);
                fprintf(in," end if; \n");
                vv[0]='1'; vv[1]='1';
                for (j=2; j<nc;j++)
                vv[j]='0';
                fprintf(in,"                    if (DELAY=\"%s\") then\n",vv);
                fprintf(in," NEW_CYCLE <='1';\n");
                fprintf(in," else \n");
                fprintf(in," NEW_CYCLE <='0';\n");
                fprintf(in," end if; \n");
```

```
                    fprintf(in," end if; \n");
                    fprintf(in,"end if; \n");
                   fprintf(in,"END PROCESS;\n\n");
                   for (j=0; j<(qt*32-n);j++)
                     vvv[j]='0';
                   if ( (qt*32-n) == 0)
                      fprintf(in,"DATA <=");
                  else
                      fprintf(in,"DATA <=\"%s\" &",vvv);
                  for (j=1; j<n;j++)
                      fprintf(in," S_Q%d(%d) &",j,nc-1);
                      fprintf(in," S_Q%d(%d); \n",j,nc-1);
                      fprintf(in,"ENABLE <= DELAY(%d) and not STALL;\n",nc-1);
fprintf(in,"\n");
fprintf(in,"END MAX2;
gen_lcell() ;
gen_comp();
gen_shiftreg() ;
gen_nb_alea() ;
gen_ahbahbtop() ;
indice=0;index=0;
for (m=1; m<n+1;m++){
       gen_gen(m,nf[m],indice);
       for (j=1;j<nf[m]+1;j++)
            indice=indice+nv[m][j];
       index=index+nvg[m];
};
indice=0; index=0;
for (i=1; i<n+1;i++){
       gen_f(i,nf[i],indice);
       for (j=1;j<nf[i]+1;j++)
           indice=indice+nv[i][j];
       index=index+nvg[i];
};
gen_des2();
fclose(in);
fclose(in3);
return 0;
}
```

ANNEXE B

Le code Vhdl d'un système composé de trois gènes

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
ENTITY Projet is
 PORT( CLK,RESET,STALL : in std_logic;
 DATA : out std_logic_vector(31 downto 0);
          ENABLE : out std_logic);
 END Projet;

ARCHITECTURE MAX2 of Projet is
 constant F11 : std_logic_vector (7 downto 0) :="11111100";
 constant F21 : std_logic_vector (7 downto 0) :="11111100";
 constant F12 : std_logic_vector (7 downto 0) :="11111100";
 constant F22 : std_logic_vector (7 downto 0) :="11111100";
 constant F13 : std_logic_vector (7 downto 0) :="01000011";
 constant F23 : std_logic_vector (7 downto 0) :="01111111";
 constant C11 : std_logic_vector (7 downto 0) :="11001100";
 constant C21 : std_logic_vector (7 downto 0) :="11111111";
 constant C12 : std_logic_vector (7 downto 0) :="11001100";
 constant C22 : std_logic_vector (7 downto 0) :="11111111";
 constant C13 : std_logic_vector (7 downto 0) :="11001100";
 constant C23 : std_logic_vector (7 downto 0) :="11111111";

COMPONENT gene1
 PORT(
 X1 : in std_logic;
 X2 : in std_logic;
 X3 : in std_logic;
 F1 : in std_logic_vector (7 downto 0) ;
 F2 : in std_logic_vector (7 downto 0) ;
 C1 : in std_logic_vector (7 downto 0) ;
 RND : in std_logic_vector (7 downto 0);
 RESULTAT : out std_logic);
 END COMPONENT ;
COMPONENT gene2
 PORT(
 X1 : in std_logic;
 X2 : in std_logic;
 X3 : in std_logic;
 F1 : in std_logic_vector (7 downto 0) ;
 F2 : in std_logic_vector (7 downto 0) ;
 C1 : in std_logic_vector (7 downto 0) ;
 RND : in std_logic_vector (7 downto 0);
 RESULTAT : out std_logic);
 END COMPONENT ;
```

```
COMPONENT gene3
 PORT(
 X1 : in std_logic;
 X2 : in std_logic;
 X3 : in std_logic;
 F1 : in std_logic_vector (7 downto 0) ;
 F2 : in std_logic_vector (7 downto 0) ;
 C1 : in std_logic_vector (7 downto 0) ;
 RND : in std_logic_vector (7 downto 0);
 RESULTAT : out std_logic);
 END COMPONENT ;
 COMPONENT shiftreg
 PORT( clk : in std_logic;
 reset : in std_logic;
 stall : in std_logic;
 clear : in std_logic;
 shift : in std_logic;
 q : out std_logic_vector(2 downto 0));
 END COMPONENT ;
 COMPONENT nb_alea
 PORT( CLK : in std_logic;
 RST : in std_logic;
 CHIF : out std_logic_vector(127 downto 0));
 END COMPONENT ;
 SIGNAL S_RND : std_logic_vector(127 downto 0);
 SIGNAL S_X1, S_X2 , S_X3 : std_logic;
 SIGNAL S_R1, S_R2, S_R3 : std_logic;
 SIGNAL S_Q1, S_Q2, S_Q3 : std_logic_vector(2 downto 0);
 SIGNAL DELAY : std_logic_vector(2 downto 0);
 SIGNAL R_vec0, R_vec1, R_vec2 : std_logic_vector(2 downto 0);
 SIGNAL NEW_CYCLE, IN_CYCLE : std_logic;
begin

NB1:nb_alea

 PORT MAP (
          CLK => CLK,
          RST => RESET,
          CHIF => S_RND);
G1:gene1

 PORT MAP (
          X1 => S_X1,
          X2 => S_X2,
          X3 => S_X3,
          F1 => F11,
          F2 => F21,
          C1 => C11,
          RND => S_RND(7 downto 0),
          RESULTAT => S_R1);
```

```
G2:gene2

 PORT MAP (
          X1 => S_X1,
           X2 => S_X2,
           X3 => S_X3,
           F1 => F12,
           F2 => F22,
           C1 => C12,
           RND => S_RND(15 downto 8),
           RESULTAT => S_R2);

G3:gene3

 PORT MAP (
          X1 => S_X1,
          X2 => S_X2,
           X3 => S_X3,
           F1 => F13,
           F2 => F23,
           C1 => C13,
           RND => S_RND(23 downto 16),
           RESULTAT => S_R3);

SHIFT1:shiftreg

 PORT MAP (
         clk => CLK,
          reset => RESET,
          stall => STALL,
          clear => NEW_CYCLE,
          shift => S_X1,
          q => S_Q1);

SHIFT2:shiftreg

 PORT MAP (
         clk => CLK,
          reset => RESET,
          stall => STALL,
          clear => NEW_CYCLE,
          shift => S_X2,
          q => S_Q2);

SHIFT3:shiftreg

 PORT MAP (
          clk => CLK,
          reset => RESET,
          stall => STALL,
          clear => NEW_CYCLE,
          shift => S_X3,
          q => S_Q3);
```

```
PROCESS(CLK,RESET)
begin
   if (RESET='0') then
         S_X1 <='1';
          S_X2 <='1';
          S_X3 <='1';
   elsif (CLK'event and CLK='1') then
         if (STALL = '0') then
               if (NEW_CYCLE = '1') then
                     S_X1 <= S_RND(127);
                     S_X2 <= S_RND(126);
                     S_X3 <= S_RND(125);
               else
                     S_X1 <= S_R1;
                     S_X2 <= S_R2;
                     S_X3 <= S_R3;
               end if;
         end if;
  end if;
END PROCESS;
R_vec0<= S_X1 & S_X2 & S_X3;
R_vec1<= S_Q1(0) & S_Q2(0) & S_Q3(0);
R_vec2<= S_Q1(1) & S_Q2(1) & S_Q3(1);
PROCESS(CLK,RESET)
begin
        if (RESET='0') then
               DELAY <= (others=>'0');
               IN_CYCLE <='0';
              NEW_CYCLE <='0';
        elsif (CLK'event and CLK='1') then
               if (STALL = '0') then
                     if (IN_CYCLE='1') then
                           if (DELAY="100") then
                                 IN_CYCLE <='0';
                                 DELAY <=(others=>'0');
                            else
                                  DELAY(2)<= DELAY(1);
                                 DELAY(1)<= DELAY(0);
                                 DELAY(0)<= '0';
                           end if;
                     elsif ( R_vec0 = R_Vec1) then
                            DELAY(0)<='1';
                           DELAY(1)<='1';
                            IN_CYCLE<='1';
                     elsif ( R_vec0 = R_Vec2) then
                           DELAY(0)<='1';
                           DELAY(1)<='1';
                           DELAY(2)<='1';
                           IN_CYCLE<='1';
                     else
                           DELAY(2)<= DELAY(1);
                           DELAY(1)<= DELAY(0);
                           DELAY(0)<= '0';
                     end if;
```

```
                        if (DELAY="110") then
                                NEW_CYCLE <='1';
                        else
                                NEW_CYCLE <='0';
                        end if;
                end if;
        end if;
END PROCESS;

DATA <="00000000000000000000000000000000" & S_Q1(2) & S_Q2(2) & S_Q3(2);
ENABLE <= DELAY(2) and not STALL;
END MAX2;
```

ANNEXE C

Les résultats obtenus à partir d'un modèle composé de 10 gènes, avec 2 prédicteurs par gène et 4 variables par prédicteur.

```
**********************************************************************
** Ce Fichier contient tous les cycles attracteurs non dupliqués ainsi que      **
** leurs occurrences                                                            **
**********************************************************************
** la taille du cycle est 2, son occurrence est : 1                            **
** 00000000000000000000000001011101                                           **
** 00000000000000000000000000011111                                           **
** 00000000000000000000000001011101                                           **
**********************************************************************
** la taille du cycle est 4, son occurrence est : 1                            **
** 00000000000000000000001100000101                                           **
** 00000000000000000000001010000101                                           **
** 00000000000000000000000010001001                                           **
** 00000000000000000000000110000000                                           **
** 00000000000000000000001100000101                                           **
**********************************************************************
** la taille du cycle est 3, son occurrence est : 1                            **
** 00000000000000000000000000011111                                           **
** 00000000000000000000000001011101                                           **
** 00000000000000000000000011011111                                           **
** 00000000000000000000000000011111                                           **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 1                            **
** 00000000000000000000001101001010                                           **
** 00000000000000000000001101001010                                           **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 747                          **
** 00000000000000000000000010011101                                           **
** 00000000000000000000000010011101                                           **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 56                           **
** 00000000000000000000001100000001                                           **
** 00000000000000000000001100000001                                           **
**********************************************************************
** la taille du cycle est 3, son occurrence est : 1                            **
** 00000000000000000000000110001101                                           **
** 00000000000000000000001010011100                                           **
** 00000000000000000000000111011100                                           **
** 00000000000000000000000110001101                                           **
**********************************************************************
```

```
**********************************************************************
** la taille du cycle est 2, son occurrence est : 3                  **
** 00000000000000000000001001001111                                  **
** 00000000000000000000001001000111                                  **
** 00000000000000000000001001001111                                  **
**********************************************************************
** la taille du cycle est 2, son occurrence est : 27                 **
** 00000000000000000000000100011101                                  **
** 00000000000000000000000010111101                                  **
** 00000000000000000000000100011101                                  **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 10                 **
** 00000000000000000000000001011111                                  **
** 00000000000000000000000001011111                                  **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 18                 **
** 00000000000000000000001101010011                                  **
** 00000000000000000000001101010011                                  **
**********************************************************************
** la taille du cycle est 3, son occurrence est : 3                  **
** 00000000000000000000000000011111                                  **
** 00000000000000000000000001011101                                  **
** 00000000000000000000000011011111                                  **
** 00000000000000000000000000011111                                  **
**********************************************************************
** la taille du cycle est 2, son occurrence est : 117                **
** 00000000000000000000000010111101                                  **
** 00000000000000000000001100011101                                  **
** 00000000000000000000000010111101                                  **
**********************************************************************
** la taille du cycle est 1, son occurrence est : 23                 **
** 00000000000000000000000001100100                                  **
** 00000000000000000000000001100100                                  **
**********************************************************************
** la taille du cycle est 3, son occurrence est : 1                  **
** 00000000000000000000000111011111                                  **
** 00000000000000000000000000111100                                  **
** 00000000000000000000000111001111                                  **
** 00000000000000000000000111011111                                  **
**********************************************************************
** la taille du cycle est 2, son occurrence est : 1                  **
** 00000000000000000000001000101101                                  **
** 00000000000000000000000101001100                                  **
** 00000000000000000000001000101101                                  **
**********************************************************************
```