

# Projet 2021

Tom Redon

May 2021

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Notions d'Architecture nécessaires</b>	<b>3</b>
<b>3</b>	<b>Génération des graphes</b>	<b>5</b>
3.1	Génération générale des graphes et algorithme de Tarjan . . . . .	5
3.2	Parcours en profondeur et signature des graphes . . . . .	6
<b>4</b>	<b>Chaînes de Markov</b>	<b>10</b>
<b>5</b>	<b>Calcul de probabilités stationnaires</b>	<b>12</b>
<b>6</b>	<b>Résultats</b>	<b>16</b>
<b>7</b>	<b>Simulateur</b>	<b>19</b>

# 1 Introduction

(A remplir)

## 2 Notions d'Architecture nécessaires

(Repréciser les pipelines et les prédicteurs, faire des rappels d'Architecture ✓)

(Préciser la raison des if dans le pipeline (il faut attendre le résultat du test pour enchaîner) ✓)

Aujourd'hui, la plupart des processeurs sont pipelinés. Cela signifie que chaque instruction est séparée en plusieurs étapes et le processeur peut ainsi exécuter plusieurs instructions en parallèle, sans avoir nécessairement besoin d'attendre la fin de l'exécution de la précédente.

Le processeur peut donc recevoir différents types d'instructions et fera en sorte d'optimiser l'exécution. On s'intéressera principalement à l'exécution des instructions conditionnelles ("if"). Ces instructions conditionnelles permettent de pouvoir "sauter" à une certaine partie de code en fonction de la valeur de la condition lue dans l'instruction. Le saut effectué est une redirection de la lecture du code vers une ligne du code assembleur, précisée par l'instruction conditionnelle. Elles sont aussi particulières car elle demande au pipeline d'attendre la fin de leur exécution avant de pouvoir continuer. On peut donc facilement se retrouver avec un pipeline en attente à cause d'instructions conditionnelles trop longues à résoudre.

Pour essayer d'optimiser l'exécution de ces instructions, le processeur essaye de réaliser des prédictions pour savoir s'il doit prendre le saut ou non. Pour pouvoir faire ces prédictions, des prédicteurs de branchements ont été développés. Il existe de nombreux types de prédicteurs de branchements. Il existe par exemple les prédicteurs statiques ou bien encore les prédicteurs dynamiques.

Les prédicteurs statiques ne se servent pas des informations récoltées pendant l'exécution du code précédent. Il aura donc des prédictions pré-établies. Par exemple, répondre à tous les branchements qu'ils sont pris. De ce fait, leur exécution est rapide, mais leur taux d'erreur de prédiction est relativement élevé.

Le modèle de prédicteur dynamique a été établi pour essayer d'obtenir un taux d'erreur de prédiction plus restreint. Il se sert des évaluations des instructions précédentes du code pour prendre ses décisions et sera souvent plus précis. Il existe différents niveaux de prédicteurs dynamiques.

On va régulièrement parler de prédicteurs à  $k$ -bits,  $k \in \mathbb{N}$ , où les  $k$  bits représentent le nombre d'information retenue. Le nombre d'états de l'automate sera  $2 * k$ . Ainsi, on pourra voir des modèles assez simples n'utilisant que la prédiction précédente pour établir la nouvelle et qui seront considérés comme des prédicteurs à 1-bit. Les modèles les plus répandus sont les prédicteurs à 2-bits, et plus précisément le compteur saturé qui permet de répondre très efficacement quand le cas étudié prend des conditions dont les réponses sont identiques pendant

un certain nombre de prédictions. On peut aussi parler du prédicteur nommé le flip-on-consecutive qui permet d'effectuer une transition de prédiction plus simplement lorsque les résultats des prédictions précédentes ne sont pas obligatoirement identiques entre elles lorsqu'elles se suivent. Vous pouvez observer un exemple de ces deux prédicteurs à 2-bits dans le schéma suivant.

(Créer deux figures différentes ✓)

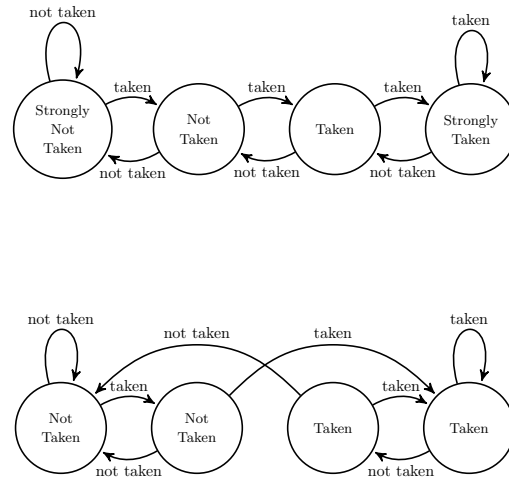


Figure 1: Deux prédicteurs à 2-bits différents (haut : compteur saturé, bas: flip-on-consecutive).

Pour représenter ces différents prédicteurs dynamiques, on souhaite étudier et donc générer les graphes à  $2 * k$ -états,  $k \in \mathbb{N}$ , fortement connexes et non-isomorphes entre eux.

### 3 Génération des graphes

Pour produire ces graphes, plusieurs méthodes ont été appliquées pendant l'avancée du projet. La première n'a pas été sélectionnée car elle était largement moins performante que la seconde. Vous pouvez donc continuer votre lecture après l'exemple.

(Repréciser pourquoi une des méthodes est moins bonne que l'autre et bien le préciser en premier lieu ✓)

#### 3.1 Génération générale des graphes et algorithme de Tarjan

En premier lieu, tous les graphes à  $k$ -états,  $k \in \mathbb{N}$ , étaient générés à partir de combinaisons établies pour représenter les différentes destinations de chaque arête.

Ainsi, pour un graphe à  $k$ -états, on a  $2 * k$  arêtes (Une arête "Taken" et une arête "Not Taken" par état).

On doit déterminer, pour chacune, l'état vers lequel elle se dirige.

Pour représenter cela, on générait des combinaisons de taille  $2 * k$  et dont chaque valeur pouvait appartenir entre 0 et  $k - 1$ .

Ces combinaisons étaient ensuite permutées de toutes les façons possibles pour pouvoir générer tous les graphes à  $k$ -états.

**Exemple 1** Pour un graphe à 4-états, on pouvait par exemple avoir la combinaison suivante :

$$\text{Combinaison} = (0 \quad 3 \quad 3 \quad 2 \quad 1 \quad 2 \quad 0 \quad 1)$$

On peut donc le lire de la façon suivante :

$\forall n \in [0, k-1], k \in \mathbb{N}, \text{Combinaison}[2*n]$  désigne l'état-destination de la branche "Not Taken".

$\forall n \in [0, k-1], k \in \mathbb{N}, \text{Combinaison}[2*n+1]$  désigne l'état-destination de la branche "Taken".

Cette combinaison produira le graphe suivant :

(Ajouter un ref et mettre l'image dans un objet flottant ✓)

On pourra ensuite permuer la combinaison pour obtenir d'autres graphes.

Cette solution n'étant pas optimale en terme de performance, il a été décidé de générer toutes les combinaisons dont les graphes découlants seraient fortement connexes et n'ayant pas déjà été générés précédemment.

On transforme ensuite ces combinaisons en graphes, puis on vérifie grâce à l'algorithme de Tarjan que les graphes ne forment bien qu'une seule composante fortement connexe chacun.

Graphe 678 :  
Score : 0.2853981633974484

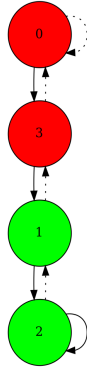


Figure 2: Premier graphe

(A modifier ✓)

L'algorithme de Tarjan étant un algorithme connu, il ne sera pas détaillé de nouveau ici. Vous pouvez cependant retrouver une description exhaustive de son but et de son implémentation ici : [1].

(A modifier ✓)

### 3.2 Parcours en profondeur et signature des graphes

(Donner un exemple pour le parcours en profondeur ✓)

(Définir les signatures pour les graphes ✓)

(Préciser les complexités)

(Faire un ensemble plus général, et en profiter pour repréciser l'exemple ✓)

(Préciser l'ancienne version et séparer toutes les méthodes dans des sous-parties)

On effectue par la suite un parcours en profondeur avec étiquetage des chemins, pour vérifier efficacement l'isomorphisme entre les graphes générés.

La signature d'un graphe est une étiquette représentant la disposition de ses états et de ses arêtes, qui aura une formulation bien précise afin de pouvoir identifier deux graphes qui auraient le même type de disposition des états, sans pour autant avoir exactement la même numérotation d'état.

Nous verrons dans l'exemple suivant comment déterminer les signatures d'un graphe.

Le principe du parcours en profondeur est donc le suivant.

Pour chaque graphe ayant été vérifié par l'algorithme de Tarjan :

- on établit une étiquette pour le graphe courant en partant du premier état (généralement l'état 0).
- on compare cette étiquette à celles précédemment enregistrées dans un ensemble, si elle n'est pas dans l'ensemble, on enregistre les k signatures existantes

pour le graphe (les autres signatures sont déterminées en commençant à partir des autres états). Si elle est dans l'ensemble, alors on passe à un autre graphe.

**Exemple 2** *En prenant donc en considération le graphe représenté précédemment (2), on pourra facilement observer la génération de la signature du graphe. On va tout d'abord observer l'état 0.*

*On crée plusieurs listes qui nous seront utiles pour la suite :  
En premier lieu, on réalise une première exploration du graphe pour marquer l'ordre de rencontre du graphe.*

*Pour cela, on va tout d'abord marquer l'état comme visité, puis on va ajouter ce dernier dans une liste qui référencera les états dans leur ordre de découverte, que l'on nommera la liste "order". Par la suite, on continuera l'exploration dans les états voisins qui n'ont pas encore été visités. Quand tous les états sont identifiés comme visités, on arrête l'exploration.*

*En partant de l'état 0, on obtiendra donc la liste "order" suivante :*

[0, 3, 1, 2]

*On effectue un second parcours, en utilisant cette liste "order" pour établir l'étiquette.*

*Pour chaque état dans la liste "order", on va regarder ses voisins dans l'ordre suivant :*

*En premier lieu, on observera le voisin auquel on peut accéder par l'arête "Not Taken", puis celui lié par l'arête "Taken". On construit ensuite la signature, en ajoutant à chaque fois la position dans la liste "order" de l'état voisin que l'on observe actuellement.*

*Ici avec la liste précédemment établie, on obtiendrait la signature suivante :*

"01021323"

*Cette chaîne de caractères est donc l'une des quatre signatures possibles pour le graphe étudié.*

Une fois que les graphes sont validés comme étant fortement connexes et non-isomorphes entre eux, on peut les considérer comme des graphes de chaînes de Markov.

(Rajouter exemple de comparaison entre 2 graphes ✓)

**Exemple 3** *On peut exposer un second exemple de graphe :*

Graphe 1 :  
Score : 0.2853981633974484

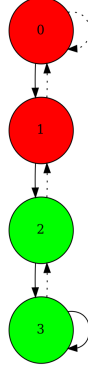


Figure 3: Second graphe

*On peut donc établir toutes les signatures des graphes tels que :*

*Graphe1 : "01021323", "12100323", "13202103", "10203132"*

*Graphe2 : "01021323", "12100323", "13202103", "10203132"*

*On peut donc voir que la signatures des graphes, il existe au moins une signature identique. Le second graphe est donc rejeté par l'algorithme et ne sera pas utilisé.*

(Rajouter un exemple avec un graphe étant invalidé par isomorphisme mais n'étant pas identique)

Les deux graphes étudiés dans l'exemple précédent sont quasiment identiques, il est très simple d'observer leur isomorphisme visuellement. On peut montrer un autre exemple d'invalidation par isomorphisme avec un graphe plus différent.

**Exemple 4** *On peut exposer un troisième exemple de graphe :*

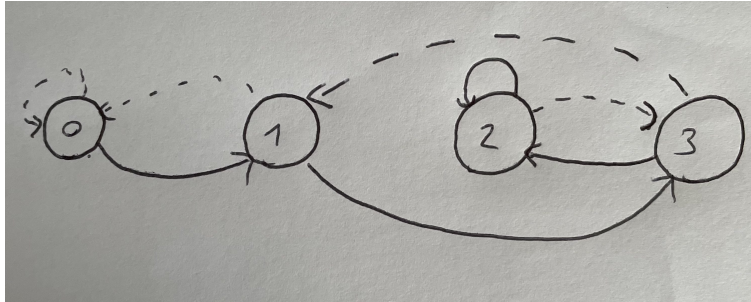


Figure 4: Troisième graphe, visuellement différent



*On peut observer que le graphe est tout d'abord différent visuellement du graphe 2. On rappelle que les signatures du graphe cité, sont :*

*"01021323" , "12100323" , "13202103" , "10203132"*

*Établissons les signatures du graphe 4. On aura donc les signatures suivantes :*

*"01021323" , "12100323" , "13202103" , "10203132"*

*On peut observer qu'il existe au moins une signature en commun entre les deux graphes, on peut donc dire que les graphes sont isomorphes et que le graphe 4 est rejeté.*

Après avoir sélectionné les graphes, on va pouvoir les transformer en chaînes de Markov et les étudier plus en profondeur.

## 4 Chaînes de Markov

- (Changer la lettre petit p dans  $p_{i,j}$  en  $\pi$  ✓)
- (Rajouter la définition (pas dans une boîte) de ce qu'est une chaîne de Markov : (In simpler terms ...) ✓)
- (Enlever la première définition ✓)
- (Faire une seule définition propre : parler du graphe de transition et de la matrice ✓)
- (Changer l'ordre des paragraphes, tout retravailler ✓)

Après avoir généré tous les graphes, nous associons des probabilités aux arêtes pour les transformer en chaînes de Markov. Tout d'abord nous devons établir quelques définitions avant de continuer. On peut tout d'abord définir une chaîne de Markov comme étant un processus qui peut effectuer chaque prédiction en ne se basant que sur l'état courant. Ces prédictions sont aussi bonnes que des prédictions réalisées en connaissant tout l'historique. [4]

**Définition 1** *Etant donné une chaîne de Markov défini sur l'ensemble d'états  $E$ , alors le nombre  $\mathbb{P}(X_1 = j \mid X_0 = i)$  est appelé probabilité de transition de l'état  $i$  à l'état  $j$  en un pas, ou bien probabilité de transition de l'état  $i$  à l'état  $j$ , s'il n'y a pas d'ambiguïté.*

*On note souvent ce nombre  $\pi_{i,j} : \pi_{i,j} = \mathbb{P}(X_1 = j \mid X_0 = i)$ .*

*La famille de nombres  $P = (\pi_{i,j})_{(i,j) \in E^2}$  est appelée matrice de transition.*

*Le graphe  $G$  d'une chaîne de Markov est un graphe orienté défini à partir de l'espace d'états  $E$  et de la matrice de transition  $P = (\pi_{i,j})$  avec  $(i,j) \in E^2$  de cette chaîne de Markov :*

- les sommets de  $G$  sont les éléments de  $E$ ,
- les arêtes de  $G$  sont les couples  $(i,j) \in E^2$  vérifiant  $\pi_{i,j} > 0$ .

- (Faire apparaître p (les probabilités) ✓)
- (Compléter les définitions pour préciser les matrices spécifiques utilisées ✓)
- (Changer l'ordre avec la proposition ✓)

Dans notre cas, on observe des instructions conditionnelles, dont la probabilité que la condition soit réalisée est  $p$ , tel que  $p \in [0, 1]$ . Cela représente la probabilité réelle d'aller d'un état au suivant en empruntant la branche "Taken".

On peut donc établir que  $\pi_{i,j}$  pourra avoir comme valeur :

- $\pi_{i,j} = p$ , pour les arêtes "Taken".
- $\pi_{i,j} = 1 - p$ , pour les arêtes "Not Taken".

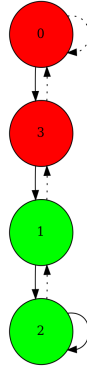
On peut en profiter pour établir la proposition suivante qui nous sera utile lors de l'utilisation de la matrice de transition pour les calculs futurs :

**Proposition 1** *La matrice de transition  $P = (p_{i,j})_{(i,j) \in E^2}$  est stochastique : la somme des termes de n'importe quelle ligne de  $P$  donne toujours 1.*

On considérera donc que ces graphes de chaînes de Markov sont représentés par des matrices de transition à l'état initial 0.

En ayant établi ces matrices et les probabilités de transitions possibles, on pourra calculer les probabilités stationnaires  $\theta_i$  pour chaque état  $i \in E$ , lié au graphe G.

Graphe 678 :  
Score : 0.2853981633974484



**Exemple 5** Servons nous donc du graphe ci-dessus, pour donner un exemple de la matrice d'adjacence que l'on obtiendrait :

$$P = \begin{pmatrix} 1-p & 0 & 0 & p \\ 0 & 0 & p & 1-p \\ 0 & 1-p & p & 0 \\ 1-p & p & 0 & 0 \end{pmatrix}$$

## 5 Calcul de probabilités stationnaires

(Introduire probabilité stationnaire ✓)

Les probabilités stationnaires  $\theta_i$  pour chaque état  $i \in E$ , lié au graphe  $G$  représentent la fraction du temps passé en chaque état  $i$ , asymptotiquement. Il existe deux façons de calculer des probabilités stationnaires. On peut soit étudier la probabilité de se retrouver dans chaque état après chaque changement :

En considérant  $X^{(n)}$ , le vecteur de probabilité. La  $i$ -ème composante désigne la probabilité d'être dans l'état  $i$  à l'instant  $n$ , pour  $n \in \mathbb{N}$ . On définit  $k$  un état précédent, soit  $k \in [0, n]$  : (Quantifier les  $X$ ,  $k$  et  $n$  ✓)

(Revoir la présentation pour la simplifier ✓)

$$X^{(n)} = X^{(n-k)} P^k$$

(Notifier l'exemple précédent avec un label ✓)

**Exemple 6** En reprenant l'exemple ci-dessus (5), on peut choisir  $X$ , telle que l'on commence à l'état 0. On aura donc :

$$X^{(0)} = (1 \quad 0 \quad 0 \quad 0)$$

On pourra donc calculer les probabilités d'être dans les prochains états à la prochaine prise de branche :

$$\begin{aligned} X^{(1)} &= X^{(0)} P = (1 \quad 0 \quad 0 \quad 0) \begin{pmatrix} 1-p & 0 & 0 & p \\ 0 & 0 & p & 1-p \\ 0 & 1-p & p & 0 \\ 1-p & p & 0 & 0 \end{pmatrix} \\ \Leftrightarrow X^{(1)} &= (1-p \quad 0 \quad 0 \quad p) \end{aligned}$$

On a donc calculé pour chaque état  $i$ , la probabilité de se retrouver à l'intérieur, en suivant une arête du graphe.

On peut considérer avec la loi des grands nombres que l'on peut établir, tel qu'en considérant le vecteur  $X$  et la variable  $n$  établis précédemment :

$$q = \lim_{n \rightarrow \infty} X^{(n)}$$

Considérons donc la matrice  $P$  (établie dans la section précédente), tel qu'il existe un vecteur  $q$  qui peut vérifier l'égalité suivante :

$$qP = q$$

Introduisons la matrice identité  $I$ , qui nous permettra d'établir l'égalité suivante :

$$\begin{aligned} \Leftrightarrow qP &= qI \\ \Leftrightarrow q(I - P) &= 0 \end{aligned}$$

**Exemple 7** En reprenant l'exemple ci-dessus, on peut établir la matrice  $q$ , telle que :

$$q = (\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3)$$

On aura donc :

$$\begin{aligned} q(I - P) &= (\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3) \left( \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1-p & 0 & 0 & p \\ 0 & 0 & p & 1-p \\ 0 & 1-p & p & 0 \\ 1-p & p & 0 & 0 \end{bmatrix} \right) \\ &\Leftrightarrow (\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3) \begin{pmatrix} p & 0 & 0 & -p \\ 0 & 1 & -p & p-1 \\ 0 & p-1 & 1-p & 0 \\ p-1 & -p & 0 & 1 \end{pmatrix} \\ &\Leftrightarrow ((-1+p)\theta_3 + p\theta_0 \quad -p\theta_3 + \theta_1 + (-1+p)\theta_2 \quad -p\theta_1 + (1-p)\theta_2 \quad \theta_3 - p\theta_0 + (-1+p)\theta_1) \end{aligned}$$

On obtiendra donc le système d'équation suivant :

$$\begin{cases} (-1+p)\theta_3 + p\theta_0 = 0 \\ -p\theta_3 + \theta_1 + (-1+p)\theta_2 = 0 \\ -p\theta_1 + (1-p)\theta_2 = 0 \\ \theta_3 - p\theta_0 + (-1+p)\theta_1 = 0 \\ \theta_0 + \theta_1 + \theta_2 + \theta_3 = 1 \end{cases}$$

Ce qui établira les prochaines égalités :

$$\begin{cases} \theta_0 = \frac{1-3p+3p^2-p^3}{1-2p+2p^2} \\ \theta_1 = \frac{p^2-p^3}{1-2p+2p^2} \\ \theta_2 = \frac{p^3}{1-2p+2p^2} \\ \theta_3 = \frac{(-1+p)^2 p}{1-2p+2p^2} \end{cases}$$

(Faire une transition et repréciser le but des calculs ✓)

On a donc obtenu des valeurs pour chaque état de leur probabilité stationnaire. Cela nous permet donc de savoir la probabilité que l'on a de se retrouver dans l'état  $i$  après les  $n$  premiers pas dans la chaîne de Markov. On pourra donc, en fonction de ces valeurs, facilement déduire si les états doivent être "Taken" ou "Not Taken". Plus précisément, pour déterminer le statut des états, on appliquera les calculs d'intégrales suivants, pour tous les états  $\pi_i$  du graphe donné :

$$\begin{aligned} x &= \int_0^1 (\theta_i * (1-p)) dp \\ y &= \int_0^1 (\theta_i * p) dp \end{aligned}$$

(Reformuler et préciser ✓)

Le  $x$  équivaut à prendre l'arête "Not Taken" pour toutes les valeurs de  $p$  possibles depuis l'état  $i$ . Le  $y$  équivaut à prendre l'arête "Taken" pour toutes les valeurs de  $p$  possibles depuis l'état  $i$ . On choisira la valeur inférieure entre les deux intégrales calculées. Cela représentera donc la prise de branche qui diminuera au maximum l'erreur de prédictions possible pour l'état. On pourra ensuite étiqueter chaque état comme étant "Taken" ou "Not Taken".

**Exemple 8** *En reprenant les calculs effectués précédemment, on obtient :  
Les calculs d'intégrales de  $\pi_0$  :*

$$\begin{aligned} x &= \int_0^1 \left( \frac{1-3p+3p^2-p^3}{1-2p+2p^2} * (1-p) \right) dp = 0.27400 \\ y &= \int_0^1 \left( \frac{1-3p+3p^2-p^3}{1-2p+2p^2} * p \right) dp = 0.08333 \end{aligned}$$

*Comme  $x \geq y$ , alors l'état est "Not Taken" car cela minimise la probabilité de mispredictions.*

*Les calculs d'intégrales de  $\pi_1$  :*

$$\begin{aligned} x &= \int_0^1 \left( \frac{p^2-p^3}{1-2p+2p^2} * (1-p) \right) dp = 0.05937 \\ y &= \int_0^1 \left( \frac{p^2-p^3}{1-2p+2p^2} * p \right) dp = 0.08333 \end{aligned}$$

*Comme  $x \leq y$ , alors l'état est "Taken".*

*Les calculs d'intégrales de  $\pi_2$  :*

$$\begin{aligned} x &= \int_0^1 \left( \frac{p^3}{1-2p+2p^2} * (1-p) \right) dp = 0.08333 \\ y &= \int_0^1 \left( \frac{p^3}{1-2p+2p^2} * p \right) dp = 0.27400 \end{aligned}$$

*Comme  $x \leq y$ , alors l'état est "Taken".*

*Les calculs d'intégrales de  $\pi_3$  :*

$$\begin{aligned} x &= \int_0^1 \left( \frac{(-1+p)^2 p}{1-2p+2p^2} * (1-p) \right) dp = 0.08333 \\ y &= \int_0^1 \left( \frac{(-1+p)^2 p}{1-2p+2p^2} * p \right) dp = 0.05937 \end{aligned}$$

*Comme  $x \geq y$ , alors l'état est "Not Taken".*

Le "score" des graphes représente le taux d'erreur de prédiction d'un graphe. Pour le calculer, on additionne juste les résultats obtenus :

**Exemple 9** *On aura donc, pour ce graphe :*

$$score = 0.08333 + 0.05937 + 0.08333 + 0.05937 = 0.2854$$

(Phrase de transition ✓)

Grâce à ces scores, nous avons pu en déduire des résultats et ainsi observer différents graphes optimaux.

## 6 Résultats

En effet, après avoir réalisé ces différents calculs, le programme est capable de réaliser un classement des prédicteurs, ordonné du plus optimal à celui qui réalise le plus d'erreur de prédictions. Il faut ainsi donné au programme un  $k$ , tel que  $k \in \mathbb{N}$  et qui représentera le nombre d'états des graphes liés aux prédicteurs. On se concentrera ici sur les prédicteurs à 2-bits, soit à  $k = 4$  états. Plus précisément, nous observerons les quatres prédicteurs les plus optimaux. Pour rappel, les numérotations d'états n'ont que peu d'importance. On peut complètement modifier cette numérotation sans provoquer de réels changements dans les prédicteurs. On s'en servira uniquement pour donner plus de clarté dans les observations.

Pour commencer, le prédicteur qui effectuera le moins d'erreur de prédictions est le compteur saturé.

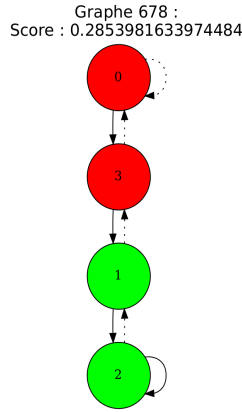


Figure 5: Compteur saturé

On peut facilement observer que le graphe représenté est identique au schéma que l'on peut observer ici, 2. Ce prédicteur fonctionne de façon optimale dans la plupart des situations.

Le second prédicteur est un mélange entre le compteur saturé et le flip-on-consecutive.



Graphe 754 :  
Score : 0.2910281534929632

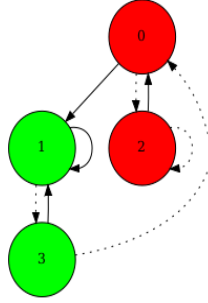


Figure 6: Hybride entre compteur saturé et flip-on-consecutive

Nous pouvons voir que la différence majeure avec le compteur saturé se fait dans les transitions entre états sans boucle. en effet, on peut même le résumer à une seule branche "Taken" de l'état "Not Taken" (l'état 0) vers l'état "Strongly Taken" (l'état 1). Ce graphe aura l'avantage de facilement transiter vers l'état "Strongly Taken" si des branches "Taken" sont souvent prises de suite.

Le troisième prédicteur est relativement identique au précédent.

Graphe 604 :  
Score : 0.29102815349296324

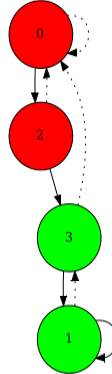


Figure 7: Second hybride entre compteur saturé et flip-on-consecutive

La différence majeure entre le second prédicteur et celui que l'on étudie actuellement est exactement l'inverse de ce que l'on a pu observer précédemment. En effet, On pourra le résumer à la branche "Not Taken" de l'état "Taken"

(l'état 3) vers l'état "Strongly Not Taken" (l'état 0). Ce graphe aura l'avantage de facilement transiter vers l'état "Strongly Not Taken" si des branches "Not Taken" sont souvent prises de suite.

Le dernier prédicteur est le flip-on-consecutive.

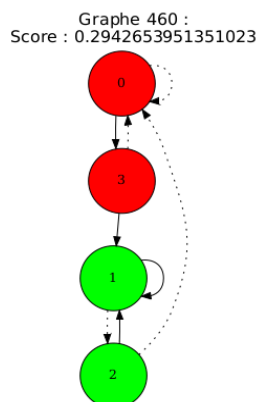


Figure 8: Flip-on-consecutive

Ce prédicteur aura l'avantage de permettre de facilement changer de type d'états entre des états "Taken" et "Not Taken" et de ne pas rester bloquer comme l'on pourrait l'être dans un compteur saturé. Il peut donc être plus optimal dans certaines situations où les résultats des instructions conditionnelles ne se suivent pas obligatoirement.

## 7 Simulateur

## References

- [1] *Algorithme de Tarjan*. URL: [https://fr.wikipedia.org/wiki/Algorithme\\_de\\_Tarjan](https://fr.wikipedia.org/wiki/Algorithme_de_Tarjan). (Visit  : 03.07.2021).
- [2] *Cha  ne de Markov*. URL: [https://fr.wikipedia.org/wiki/Cha%C3%A9ne\\_de\\_Markov](https://fr.wikipedia.org/wiki/Cha%C3%A9ne_de_Markov). (Visit  : 30.06.2021).
- [3] *Graphe d'une cha  ne de Markov*. URL: [https://fr.wikipedia.org/wiki/Graphe\\_d%27une\\_cha%C3%A9ne\\_de\\_Markov\\_et\\_classification\\_des\\_%C3%A9tats](https://fr.wikipedia.org/wiki/Graphe_d%27une_cha%C3%A9ne_de_Markov_et_classification_des_%C3%A9tats). (Visit  : 30.06.2021).
- [4] *Markov Chain*. URL: [https://en.wikipedia.org/wiki/Markov\\_chain](https://en.wikipedia.org/wiki/Markov_chain). (Visit  : 03.07.2021).