
Exploration aléatoire de modèles

Johan Oudinet

LRI (Univ. Paris-Sud – CNRS)

Bât 490

F-91405 Orsay Cedex

johan.oudinet@lri.fr

RÉSUMÉ. Cet article présente des méthodes d'explorations probabilistes qui garantissent une bonne couverture des traces du modèle quelle que soit sa topologie. Ces méthodes sont basées sur des techniques de comptage et de tirage uniforme de structures combinatoires qui dans leurs versions de bases sont très coûteuses en espace mémoire. Ce papier présente des améliorations qui permettent d'explorer uniformément des modèles encore plus gros en tirant des traces encore plus longues. Une étude de la complexité binaire, accompagnée de résultats expérimentaux de chacune des variantes possibles permet de décider quelle variante utiliser en fonction de la taille du modèle, des longueurs de traces désirées et des ressources disponibles.

ABSTRACT. We present randomized methods to explore models and ensure a good coverage of traces whatever the model topology. These methods rely on counting and uniform drawing in combinatorial structures. In their basic version, such methods are space consuming. We improve them to be able to uniformly explore larger models and draw longer traces.

MOTS-CLÉS : génération aléatoire uniforme, exploration de modèles, arithmétique flottante.

KEYWORDS: uniform random generation, model exploration, floating point arithmetic.

1. Introduction

Cet article présente des méthodes d'explorations probabilistes de modèles finis non probabilistes. Il existe plusieurs critères de couverture (états, transitions, MC/DC, chemins) en fonction des propriétés qu'on cherche à vérifier. Ici, nous nous intéressons à la couverture uniforme de chemins.

Le problème de l'explosion combinatoire est un phénomène connu de toute personne qui étudie des modèles, quelque soit leur nature. Les systèmes considérés deviennent de plus en plus complexes et leurs modèles croissent de façon exponentielle. Par exemple, la composition de modèles conduit généralement à des modèles de taille exponentielle. Ou bien, le simple fait de considérer les types de données, même en les bornant, produit là encore des modèles de très grandes tailles.

L'exploration aléatoire d'un modèle est une approche classique en simulation, mais est aussi utilisée pour le test (Dwyer *et al.*, 2007) et le model-checking (Grosu *et al.*, 2005; Hérault *et al.*, 2004).

Pour explorer aléatoirement un modèle représenté par un graphe, l'approche la plus naturelle est l'utilisation de marches aléatoires. Pour effectuer une marche aléatoire sur un graphe \mathcal{G} , il suffit de connaître tous les états du système ainsi que leurs successeurs, et d'être capable d'attribuer à chacun de ces successeurs une probabilité telle que la somme des probabilités de tous les successeurs d'un sommet soit égale à 1. Dans le cas d'une marche aléatoire *isotrope*, tous les successeurs sont équiprobables. L'algorithme 1 permet de générer un chemin d'une longueur inférieure ou égale à n .

Algorithme 1 Marche aléatoire isotrope sur un graphe

Entrées: Un graphe \mathcal{G} , un sommet initial s_0 et une longueur n

Sortie: Un chemin σ tel que $|\sigma| \leq n$

$i \leftarrow 0$

$s \leftarrow s_0$

tant que $i \neq n$ **et** $\text{succ}(s) \neq \emptyset$ **faire**

 Choisir un sommet s' uniformément parmi les successeurs de s ($\text{succ}(s)$)

$\sigma \leftarrow \sigma \cup t$ avec t la transition $s \rightarrow s'$

$i \leftarrow i + 1$

$s \leftarrow s'$

fin tant que

retourne σ

N'ayant besoin d'aucune autre connaissance que l'état courant et la liste de ses successeurs, une marche aléatoire isotrope semble être le candidat idéal pour explorer de très grands modèles. Malheureusement, la distribution de probabilité sur les chemins induite est difficile à déterminer car elle dépend de la topologie du graphe. Il peut arriver qu'une marche aléatoire isotrope soit totalement inefficace, comme pour l'exemple suivant.

Soit le graphe de la figure 1, l'espérance du nombre N de marches aléatoires isotropes à effectuer avant d'obtenir n chemins distincts (de longueur n) est de :

$$\begin{aligned}
 E(N) &= E(N_1) + E(N_2) + \dots + E(N_n) \\
 &= \frac{1}{p_1} + \frac{1}{p_2} + \dots + \frac{1}{p_n} \\
 &= 1 + 2 + \dots + 2^{n-1} \\
 &= 2^n - 1
 \end{aligned}
 \tag{1}$$

où $E(N_i)$ (resp. p_i) désigne l'espérance (resp. la probabilité) d'obtenir un nouveau chemin après avoir effectué $i - 1$ marches aléatoires isotropes distinctes. En d'autres termes, l'équation [1] montre qu'en utilisant des marches aléatoires isotropes, il faut en moyenne un temps exponentiel pour couvrir les chemins du graphe de la figure 1.

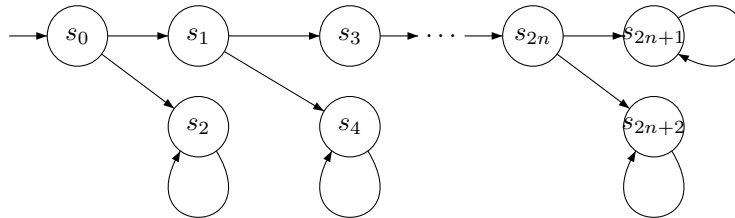


Figure 1. Un exemple de graphe pathologique pour la marche aléatoire isotrope

Ce temps exponentiel est dû au fait qu'à chaque sommet la marche aléatoire doit faire un choix entre soit un sommet qui mène à un seul chemin, soit un sommet d'où partent un nombre exponentiel de chemins. Or dans le cas d'une marche isotrope, ces deux sommets ont la même probabilité d'être tirés, favorisant ainsi la probabilité d'apparition d'un chemin au détriment d'un très grand nombre de chemins ! Si on était capable de connaître le nombre de chemins partant de chacun des sommets au moment de choisir le sommet suivant, alors on pourrait guider la marche aléatoire afin de rééquilibrer la probabilité d'apparition de tous les chemins et d'obtenir, au mieux, une distribution uniforme sur les chemins.

Cet article présente des méthodes d'explorations probabilistes qui garantissent une bonne couverture des traces du modèle quelle que soit sa topologie. Ces méthodes sont basées sur des techniques de comptage et de tirage uniforme de structures combinatoires qui dans leurs versions de bases sont très coûteuses en espace mémoire. Nous avons déjà développé des techniques de tirage modulaire qui se basent sur le tirage de chemins uniforme dans chaque composant (Gaudel *et al.*, 2008). L'objectif de cet article est d'améliorer le tirage dans chaque composant, dont la taille est supposée tenir en mémoire, afin de pouvoir tirer des chemins encore plus longs et donc d'être capable de tirer des chemins plus longs aussi avec les techniques de tirage modulaire.

La section 2 explique une méthode utilisée par Denise *et al.* (Denise *et al.*, 2004; Gouraud *et al.*, 2001) pour faire du tirage uniforme de chemins de longueur inférieure

ou égale à n , qui se base sur une méthode récursive pour compter les chemins (Flajolet *et al.*, 1994). Cet article présente, dans les sections 3 et 4, deux optimisations qui permettent de tirer des chemins encore plus longs et plus rapidement. Les résultats des premières expériences menées avec ces nouvelles méthodes sont présentés dans la section 5.

2. Tirage uniforme de chemins

Il est nécessaire d'être en mesure de compter les chemins partant d'un sommet afin d'obtenir un tirage uniforme sur ces chemins. Dans cette section, je résumerai les travaux de Denise *et al.* concernant le tirage uniforme de chemins de longueur inférieure ou égale à n . Mais avant cela, nous avons besoin d'un minimum de formalisme.

Les modèles peuvent être représentés de différentes manières selon la sémantique que l'on veut leur donner et le type d'analyse que l'on désire effectuer. Pour la suite de cet article, nous utiliserons la notion d'automate.

Définition 1. Un automate fini \mathcal{A} est une structure :

$$\mathcal{A} = \langle \mathcal{X}, \mathcal{S}, s_0, \mathcal{F}, \mathcal{T} \rangle$$

où \mathcal{X} est un alphabet d'étiquettes, \mathcal{S} un ensemble fini d'états, s_0 l'état initial, $\mathcal{F} \subseteq \mathcal{S}$ un ensemble d'états finaux, et $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{X} \times \mathcal{S}$ un ensemble de transitions.

Définition 2. Un chemin σ de longueur n dans un automate \mathcal{A} est une séquence de transitions :

$$\sigma = t_1 t_2 \cdots t_n$$

telle que : $\forall i, t_i = s_{i-1} \times a_i \times s_i$ et $s_n \in \mathcal{F}$.

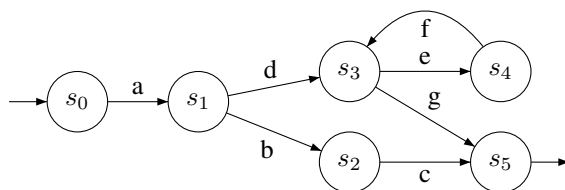


Figure 2. Un exemple d'automate

La figure 2 donne un exemple d'automate avec un seul état final (s_5). Le chemin $\sigma = s_0, a, s_1, b, s_2, c, s_5$ est un des deux chemins de longueur 3 de cet automate. Il y a en tout 6 chemins de longueur ≤ 9 .

Soit un automate \mathcal{A} et un entier n , $\mathcal{P}_n(\mathcal{A})$ (resp. $\mathcal{P}_{\leq n}(\mathcal{A})$) désigne l'ensemble des chemins de longueur n (resp. inférieure ou égale à n) dans \mathcal{A} . Lorsqu'il n'y a pas d'ambiguïté sur \mathcal{A} , on utilisera la notation réduite \mathcal{P}_n (resp. $\mathcal{P}_{\leq n}$). On peut noter que la taille de ces ensembles est généralement exponentielle par rapport à n .

La problématique est le tirage uniforme des chemins de $\mathcal{P}_{\leq n}$. Mais avant cela, nous allons nous intéresser au tirage uniforme de chemins de longueur exactement n , c'est-à-dire aux chemins qui sont dans \mathcal{P}_n . Nous verrons ensuite qu'une petite modification dans l'automate permet d'engendrer des chemins de longueurs $\leq n$, sans changer la méthode.

Supposons qu'à l'étape en cours, la marche aléatoire est sur l'état s , qui a k successeurs : s_1, s_2, \dots, s_k , et qu'il reste m étapes avant d'obtenir un chemin de longueur n . Pour garantir l'uniformité des chemins de \mathcal{P}_n , la marche aléatoire doit choisir le successeur s_i (pour $1 \leq i \leq k$) avec la probabilité suivante :

$$P(s_i) = \frac{l_{s_i}(m-1)}{l_s(m)} \quad [2]$$

avec $l_s(m)$ qui désigne le nombre de chemins de longueur m qui vont de s à un état appartenant à \mathcal{F} .

Pour obtenir un tirage uniforme il est nécessaire de calculer $l_s(m)$ pour tout $s \in \mathcal{S}$ et tout m , $0 \leq m \leq n$. Ce calcul peut être fait avec la relation de récurrence suivante :

$$\begin{cases} l_s(0) = 1 & \text{si } s \in \mathcal{F} \\ l_s(0) = 0 & \text{si } s \notin \mathcal{F} \\ l_s(i) = \sum_{s \rightarrow s'} l_{s'}(i-1) & \forall i > 0 \end{cases} \quad [3]$$

Ainsi, après un prétraitement qui consiste à stocker au préalable les valeurs $l_s(i)$ pour tout s et pour $0 \leq i \leq n$ dans un tableau à deux dimensions - appelé la table de comptage - on peut tirer uniformément des chemins de longueur n .

En ce qui concerne le prétraitement, le calcul de la table de comptage nécessite $\mathcal{O}(n \times d \times S)$ additions, où n est la longueur des chemins à tirer, S le nombre d'états de l'automate et d son degré sortant maximal. Chaque opération est effectuée en utilisant une arithmétique entière. Or, les deux opérandes, qui représentent des nombres de chemins, peuvent avoir une taille importante (en $\mathcal{O}(n)$) car la place nécessaire pour représenter un nombre est d'un ordre de grandeur logarithmique par rapport à sa valeur et le nombre de chemins de longueur n peut être exponentiel par rapport à n . Ainsi, le coût de chaque opération arithmétique est au pire en $\mathcal{O}(n)$ (Knuth, 1997) et la complexité du prétraitement en nombre d'opérations binaires est en $\mathcal{O}(n^2 \times d \times S)$. Concernant la complexité en mémoire du prétraitement, le stockage de la table de comptage requiert $\mathcal{O}(n \times S)$ grands nombres. Chaque nombre pouvant avoir une taille en $\mathcal{O}(n)$, l'occupation mémoire est en $\mathcal{O}(n^2 \times S)$.

En ce qui concerne le tirage, l'obtention d'un chemin de longueur n demande $\mathcal{O}(n \times d)$ opérations arithmétiques. Or, comme précédemment, chaque opération arithmétique étant en $\mathcal{O}(n)$, la complexité binaire du tirage d'un chemin de longueur n est en $\mathcal{O}(n^2 \times d)$. Et l'occupation mémoire nécessaire est celle de la table, $\mathcal{O}(n^2 \times S)$, plus celle d'un chemin de longueur n , $\mathcal{O}(n)$, soit une complexité mémoire en $\mathcal{O}(n^2 \times S)$ pour le tirage d'un chemin de longueur n .

Mais qu'en est-il des chemins de longueur $\leq n$? Pour les obtenir, il suffit d'ajouter un état s'_0 qui devient le nouvel état initial de \mathcal{A} , une (τ -)transition qui va de s'_0 à s_0 , et une (τ -)boucle sur s'_0 . Il est évident qu'un chemin de longueur $n + 1$ qui commence par $k + 1$ (τ -)transitions dans ce nouvel automate correspond à un chemin de longueur $n - k$ dans l'automate précédent. On vérifie aussi aisément que tout ancien chemin de longueur inférieure ou égale à n est maintenant représenté par un chemin de longueur exactement $n + 1$.

Les limites de cette méthode sont : l'espace mémoire nécessaire pour la table de comptage, qui peut être très important dès lors qu'on s'intéresse à tirer des chemins longs dans de très gros modèles, et dans une moindre mesure, le temps de génération qui peut devenir quadratique en n si les opérations sont faites avec une arithmétique entière. Les deux sections suivantes décrivent, respectivement, deux variantes de cette méthode qui peuvent être combinées pour dépasser ces limites.

3. Génération uniforme sans table

Si on regarde plus en détails la création et l'utilisation de la table de comptage dans la méthode de tirage décrite à la section précédente, on remarque les points suivants :

- La table est remplie, avec la relation de récurrence [3], des lignes 0 à n .
- Puis, à chaque génération d'un chemin de longueur n , la table est parcourue dans le sens inverse, à savoir de la ligne n à la ligne 0. Or, à chaque étape de la marche aléatoire, seules deux lignes (les lignes $i - 1$ et i) sont utiles.

Par exemple, en considérant la table de comptage du tableau 1, la marche aléatoire qui permet de générer le chemin de longueur 5 n'aura besoin que des lignes 3 et 4 lorsqu'elle sera sur l'état s_1 pour choisir avec probabilité 1 l'état s_3 .

n	sommets					
	s_0	s_1	s_2	s_3	s_4	s_5
0	0	0	0	0	0	1
1	0	0	1	1	0	0
2	0	2	0	0	1	0
3	2	0	0	1	0	0
4	0	1	0	0	0	0
5	1	0	0	0	0	0

Tableau 1. Table de comptage associée au graphe de la figure 2, pour $n = 5$

Ainsi, si on était capable de calculer, pour tout état s , $l_s(i - 1)$ à partir des $l_s(i)$, il ne serait alors plus nécessaire de garder en mémoire toute la table de comptage, mais seulement deux lignes, que l'on mettrait à jour à chaque étape de la marche aléatoire. La méthode devient :

Prétraitement : Calculer la dernière ligne (n) de la table avec l'équation [3]. Seule la dernière ligne est conservée en mémoire.

Génération de chemins : Le principe reste identique, sauf qu'à chaque étape de la marche aléatoire, on calcule la ligne précédente de la table de comptage.

Plus précisément, pour être en mesure de calculer la ligne précédente lors du tirage, nous allons formuler le problème sous une forme matricielle. Soit $A \in \mathbb{N}^{S \times S}$ la matrice de transitions de \mathcal{A} , et le vecteur $F \in \mathbb{N}^S$ défini ainsi :

$$F[i] = \begin{cases} 1 & \text{si } s_i \in \mathcal{F} \\ 0 & \text{sinon.} \end{cases} \quad [4]$$

Le vecteur $L_n = A^n F$ représente, pour chaque état, le nombre de chemins de longueur n qui finissent dans un état final. La problématique de l'inversion du système de récurrences défini par l'équation [3] est alors équivalente au problème d'algèbre linéaire suivant : Trouver un entier n_0 et une matrice $B \in \mathbb{Q}^{S \times S}$ tels que

$$\forall i \geq n_0, \quad B.A^{i+1} = A^i. \quad [5]$$

L'ajout de n_0 , l'entier minimum à partir duquel la relation est valide, est obligatoire car il peut exister un rang où il est impossible de retrouver la ligne précédente. Par exemple, si le modèle ne comporte aucun chemin de longueur supérieure ou égale à 7, alors A^7 est la matrice nulle et il est impossible de trouver A^6 à partir de A^7 . La solution à ce problème d'algèbre est composée de deux étapes :

1) Trouver le plus petit i tel que le rang de A^{i+1} soit égal au rang de A^i , ce i est en fait $n_0 - 1$.

2) Comme le rang de A^{n_0} est égal au rang de A^{n_0+1} , en notant f l'application linéaire associée à A et f_{n_0} celle associée à A^{n_0} , on a que la restriction de f à l'image de f_{n_0} est un isomorphisme (c.à-d., que la matrice C correspondante est inversible). Il suffit alors de calculer cette matrice C , de l'inverser et de revenir dans l'espace d'origine pour obtenir la matrice B recherchée.

L'obtention de A^{n_0} et de son rang demande des multiplications successives de matrices creuses d'entiers. Chaque entier est borné par $\mathcal{O}(d^{n_0})$ où d est le degré sortant maximal de \mathcal{A} ; leur taille est donc en $\mathcal{O}(n_0)$. La complexité binaire de la première étape est en $\mathcal{O}(S^3 \times n_0^2)$ (ou en $\mathcal{O}(S^2 \times n_0^2)$ si la matrice est creuse). L'opération coûteuse de la deuxième étape est l'inversion de la matrice C qui est de taille $r \times r$ (r étant le rang de A^{n_0}), ce qui donne une complexité binaire en $\mathcal{O}(r^3)$ (ou en $\mathcal{O}(r^2)$ si C est une matrice creuse).

1. Comme le rang de A^{i+1} est toujours positif ou nul et inférieur ou égal à celui de A^i , on a que n_0 existe toujours et qu'il vaut au plus S . On aurait pu choisir directement $n_0 = S$ mais la taille de l'automate étant grande, il est préférable d'essayer de trouver un n_0 beaucoup plus petit.

Ainsi, l'inversion d'un système de récurrences est polynomiale en la taille du système, S . Mais, comme pour le prétraitement quand on utilise la table de comptage, cette inversion n'a besoin d'être effectuée qu'une seule fois quelque soit le nombre de chemins tirés ensuite. Cependant, le coût de génération d'un chemin de longueur n a augmenté car le calcul de la ligne précédente de la table de comptage en utilisant les relations de récurrences définies par B nécessite $S \times d$ multiplications d'un rationnel par un grand entier. Or la complexité binaire d'une multiplication est toujours supérieure à celle d'une addition. En considérant que le coût d'une multiplication est quadratique en la taille des opérandes, soit en $\mathcal{O}(n^2)$ dans notre cas, la complexité binaire du tirage passe en $\mathcal{O}(n^3 \times d \times S)$. Mais il faut prendre en compte le gain en taille mémoire qui est très important quand on veut tirer des chemins très longs.

4. Calcul flottant

Une variante possible (Denise *et al.*, 1999) est d'éviter le coût prohibitif de chaque opération en remplaçant l'arithmétique entière par une arithmétique flottante.

L'utilisation de nombres à virgule flottante assure un temps de calcul pour chaque opération en $\mathcal{O}(p)$ où p est la précision choisie (c.à-d., la taille de la mantisse), bien inférieure à n ; le coût de chaque opération devient indépendant de n et est considéré comme une constante. Les calculs sont certes approchés, mais avec l'emploi de bibliothèques telles que MPFR (Fousse *et al.*, 2007), on a la garantie d'obtenir le réel - exprimable avec la précision choisie - le plus proche de la valeur exacte. Alain Denise et Paul Zimmermann donnent une borne théorique de la déviation maximum qui peut arriver en utilisant une arithmétique flottante certifiée. En interprétant leur théorème 4.1 avec nos notations, on a que la probabilité \tilde{p}_n d'un chemin de longueur n dévie au plus de $\mathcal{O}(n \times d \times 2^{-p})$ par rapport à la probabilité uniforme p_n . Mais en pratique, que ce soit dans les expériences réalisées par Alain Denise et Paul Zimmermann ou dans celles dans la section 5, l'erreur constatée est souvent beaucoup plus faible.

Le tableau 2 récapitule pour d fixé, ainsi que p et n_0 , les complexités (en nombre d'opérations binaires et en mémoire) obtenues en fonction de la variante utilisée : avec table ou sans table, avec calcul exact ou flottant.

5. Expériences

Dans cette section, je présente les résultats de la comparaison, en temps et en mémoire, des quatre variantes possibles. Ainsi qu'une étude de la déviation par rapport à l'uniformité pour les deux variantes qui utilisent le calcul flottant.

5.1. Implémentation et méthodologie

Le calcul du système de récurrences inverses, nécessaire pour la génération de chemins sans table de comptage, est réalisé par le processus suivant : Tout d'abord, un script SAGE (un environnement libre de calcul formel (Stein, 2007)) est créé automatiquement à partir du graphe du modèle : il commence par construire la matrice de transitions (en utilisant une représentation de matrice creuse), puis il calcule l'indice minimum à partir duquel l'inversion est possible et enfin il calcule le système de récurrences inverses à l'aide de la fonction `solve_left` de SAGE .

Toutes les expériences ont été réalisées sur un serveur *Debian 64 bits*, composé d'un processeur Intel Xeon 3GHz avec 16Gio de mémoire vive. Les graphes, au format `GraphViz` après conversion, proviennent tous de la base de modèles VLTS (Very Large Transition Systems (Garavel *et al.*, 2003)). Ces modèles correspondent à des systèmes industriels réels et leur matrice de transition est creuse (la valeur de d est petite par rapport à S). Leur nom est de la forme *vasy_X_Y*, où X est le nombre d'états divisé par 1000, et Y est le nombre de transitions divisé par 1000. Toutes les mesures ont été réalisées 10 fois et nous indiquons la valeur moyenne.

Les quatre variantes de la méthode de génération uniforme de chemins sont accessibles via une interface commune écrite en C++ (tous les codes sources sont disponibles dès maintenant sur demande et seront prochainement mis en ligne sur ma page Internet). Cette interface générique permet de passer aisément d'une variante à l'autre. J'ai utilisé plusieurs outils et bibliothèques que je mentionne ici : la commande `bcg_io` de la boîte à outils CADP (Garavel *et al.*, 2001) pour convertir les modèles de VLTS au format `GraphViz` ; plusieurs bibliothèques C++ de BOOST, BGL (Boost Graph Library (Siek *et al.*, 2000)) pour la manipulation de graphes, RANDOM (Maurer *et al.*, 2000) pour la génération de nombres aléatoires ; et les bibliothèques GMP (Granlund, 2007) et MPFR (Fousse *et al.*, 2007) pour le calcul exact et le calcul flottant, respectivement.

Méthode	Calcul	Prétraitement		Tirage	
		Temps	Espace	Temps	Espace
table	exact	$\mathcal{O}(n^2 \times S)$	$\mathcal{O}(n^2 \times S)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \times S)$
sans table	exact	$\mathcal{O}(n^2 \times S + S^2)$	$\mathcal{O}(n \times S)$	$\mathcal{O}(n^3 \times S)$	$\mathcal{O}(n \times S)$
table	flottant	$\mathcal{O}(n \times S)$	$\mathcal{O}(n \times S)$	$\mathcal{O}(n)$	$\mathcal{O}(n \times S)$
sans table	flottant	$\mathcal{O}(n \times S + S^2)$	$\mathcal{O}(S)$	$\mathcal{O}(n \times S)$	$\mathcal{O}(S)$

Tableau 2. Récapitulatif des complexités en nombre d'opérations binaires et en espace mémoire en fonction de la variante utilisée. n désigne la longueur maximale des chemins tirés et S le nombre d'états dans le système. Dans un souci de clarté, nous avons considéré comme des constantes les valeurs suivantes : le degré maximal d de l'automate, la valeur n_0 à partir de laquelle le système d'inversion est correct, et la précision p choisie pour les calculs flottants

5.2. Consommation mémoire

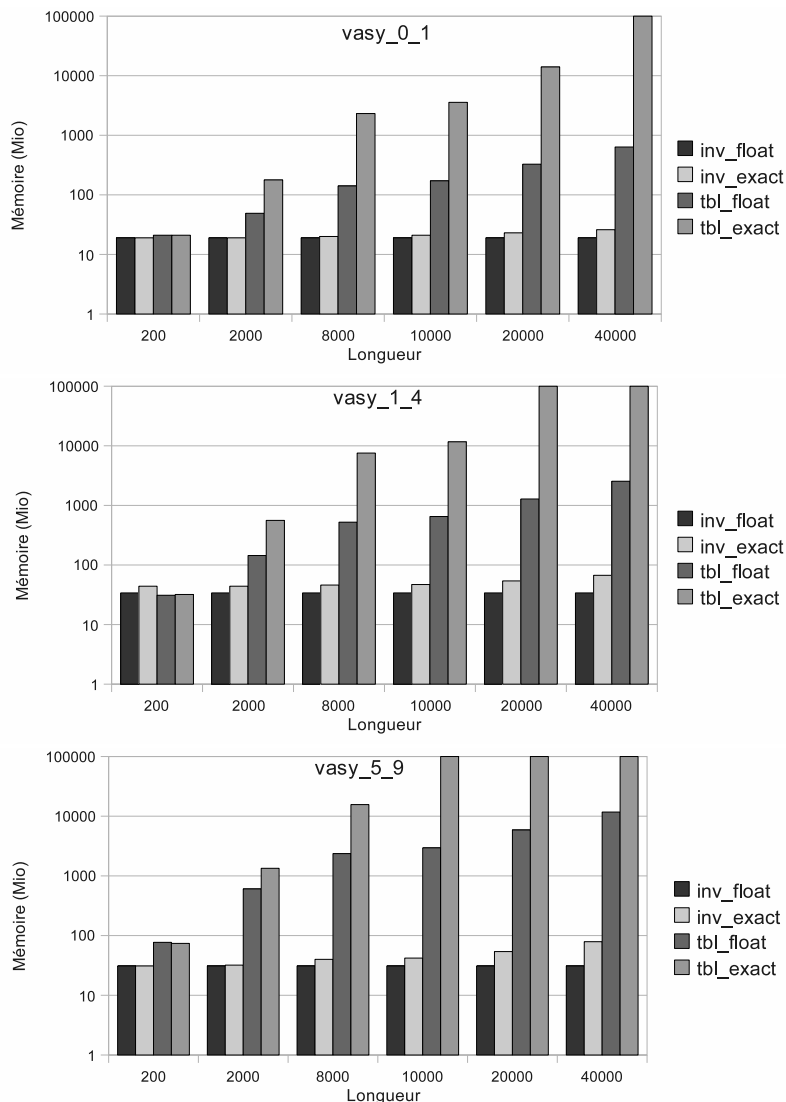


Figure 3. Comparaison de la consommation mémoire des quatre variantes en fonction de la longueur des chemins, sur trois modèles différents

La figure 3 montre la consommation mémoire nécessaire pour chacune des quatre variantes (*tbl_exact* pour la version d’origine, à savoir le calcul de la table de comptage avec une précision infinie ; *tbl_float* pour la version aussi avec table de comptage mais où tous les calculs sont faits avec une précision de 64 bits ; et enfin *inv_exact* et *inv_float* pour les versions qui ne gardent pas la table de comptage en mémoire et où les calculs sont respectivement exacts et avec une précision de 64 bits). Pour chacun

des 3 modèles, les quatre variantes ont été exécutées pour tirer des chemins dont la longueur varie entre 200 et 40000. La valeur de 100000Mio est fictive et signifie que la méthode demandait plus de mémoire que les 16Gio disponibles. Ainsi il est impossible de tirer des chemins de longueur 10000 ou plus dans le modèle vasy_5_9 avec la méthode d'origine.

À chaque fois, la version en calcul flottant nécessite moins de mémoire que son homologue en calcul exact. En ce qui concerne les versions qui ne gardent pas en mémoire la table de comptage, elles consomment toujours beaucoup moins de mémoire que les variantes avec table de comptage, à part pour de très petites longueurs où le stockage de la matrice B , permettant d'inverser les récurrences, peut être plus coûteux que la table de comptage elle-même.

5.3. Temps d'exécution

La figure 4 récapitule le temps mis par chacune des quatre variantes pour tirer 100 chemins dans les 3 mêmes modèles que précédemment. Les temps de tirage étant plus longs pour les variantes sans table de comptage, un seul chemin a été généré avec ces variantes et le temps de génération a été multiplié par 100 pour obtenir le temps de tirage de 100 chemins. De plus, la valeur fictive de 10 millions de secondes signifie que soit la méthode n'a pas pu fonctionner car elle n'avait pas suffisamment de mémoire (voir figure 3) soit l'expérience a duré plus de 24h.

Les méthodes sans table de comptage permettent de tirer des chemins très longs en utilisant peu de mémoire. En revanche, lorsqu'il est possible de garder la table de comptage en mémoire, il est préférable d'utiliser les variantes avec table qui permettent de tirer des chemins beaucoup plus rapidement.

Les méthodes en calcul flottant sont plus rapides et consomment moins de mémoire que leurs homologues en calcul exact. On pourrait penser qu'il est toujours préférable d'utiliser le calcul flottant, mais il faut faire attention à la déviation possible par rapport à l'uniformité qui pourrait dans certains cas être problématique. C'est l'étude de la section suivante.

5.4. Déviation par rapport à l'uniformité

En reprenant la notation $l_s(i)$, définie à la section 2 pour le nombre de chemins de longueur i partant du sommet s calculé à l'aide de la table de comptage avec calcul exacts, et en nommant $\tilde{l}_s(i)$ la même valeur obtenue en calcul flottant, on peut mesurer la déviation maximale, en mesurant l'erreur relative de ces deux valeurs par rapport au nombre de chemins de longueur n partant du sommet initial s_0 , par la formule suivante :

$$Err = \max_{\substack{s \in S \\ 0 < i \leq n}} \frac{|l_s(i) - \tilde{l}_s(i)|}{l_{s_0}(n)} \quad [6]$$

La figure 5 indique la valeur de Err , obtenue par la formule [6], pour les deux méthodes utilisant le calcul flottant (suffixe *tbl* pour la version avec table de comptage et suffixe *inv* pour celle sans table de comptage) sur les 3 modèles (*vasy_0_1*, *vasy_1_4*, *vasy_5_9*).

À chaque fois, l’algorithme avec table de comptage est stable numériquement avec une déviation maximale d’environ 10^{-20} . Par contre, la version sans table est instable

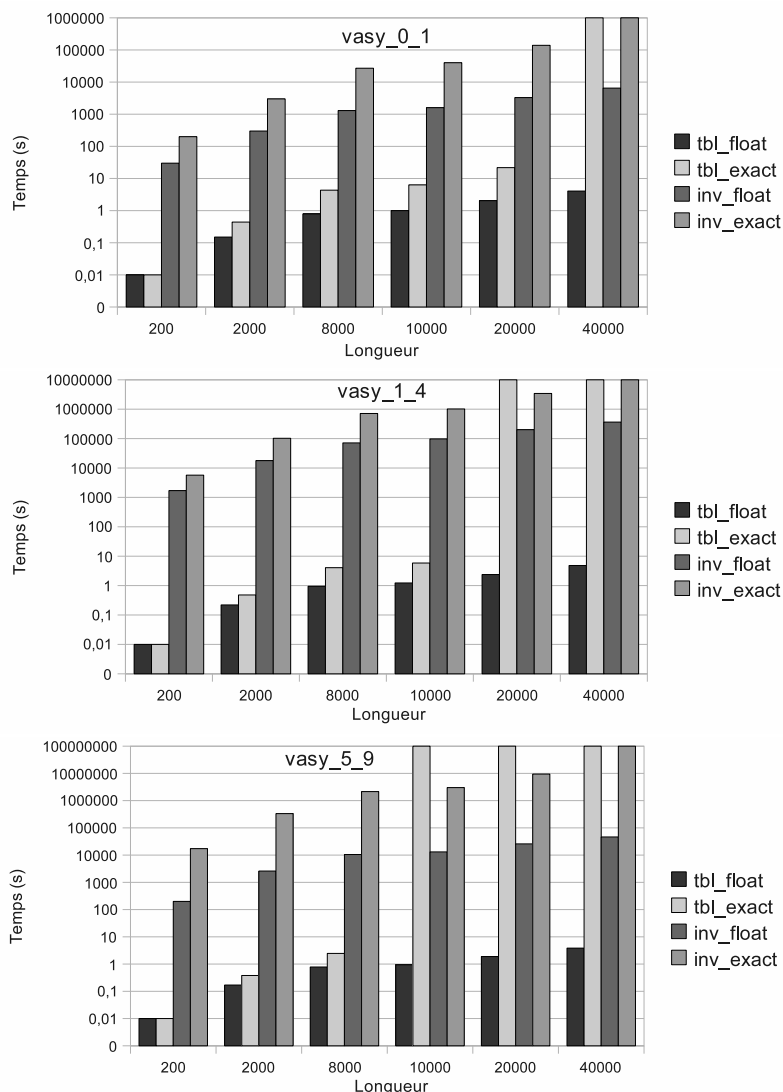


Figure 4. Temps mis par chacune des quatre variantes pour tirer 100 chemins dans chacun des 3 modèles, en fonction de la longueur des chemins

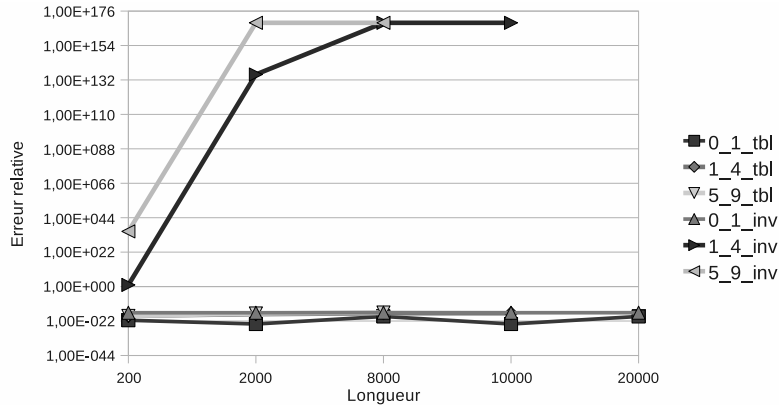


Figure 5. Mesure de l'erreur relative maximale, définie par l'équation [6], pour les deux variantes qui utilisent le calcul flottant (inv et tbl) sur les 3 modèles

numériquement : elle peut être utilisée sur le modèle vasy_0_1, mais pas sur les modèles vasy_1_4 et vasy_5_9. Cette instabilité numérique s'explique par la présence de valeurs négatives dans la matrice B qui peuvent provoquer un phénomène d'annulation entre deux nombres proches ayant pour conséquence d'aggraver les erreurs d'arrondis (Enge *et al.*, 2009).

6. Conclusions et perspectives

Nous avons développé dans cet article des méthodes qui permettent l'exploration efficace de très grands modèles. Elles réalisent une exploration *aléatoire* tout en garantissant une bonne couverture des chemins du modèle quelle que soit sa topologie, ce qui n'est pas le cas avec une marche aléatoire isotrope.

Notons que ces méthodes pourraient être étendues à la couverture des états et des transitions selon les principes donnés dans (Gaudel *et al.*, 2008; Gouraud *et al.*, 2001).

Une limite que l'on pouvait reprocher à la première version de cette méthode était la nécessité de la table de comptage. Cette table impose d'avoir un espace mémoire proportionnel au produit du nombre d'états de l'automate (S) par la longueur des chemins tirés (n), soit en $\mathcal{O}(n^2 \times S)$; ce qui est considérable dès lors qu'on s'intéresse à tirer de longs chemins dans de très grands modèles (Gaudel *et al.*, 2008). Mais, avec les améliorations introduites aux sections 3 et 4, l'espace mémoire nécessaire est désormais en $\mathcal{O}(S)$. Ce qui est excellent car on rappelle que dans le cas d'exploration modulaire, S désigne la taille du composant et non du système global.

De plus, les expériences réalisées dans la section 5 montrent que les variantes avec calcul flottant utilisent moins de mémoire et sont plus rapides qu'en calcul exact, au détriment de l'exactitude de l'uniformité sur les chemins (même si en pratique aucune différence n'a été constatée pour la variante avec table). La version sans table est utile

si les ressources disponibles ne permettent pas de stocker la table de comptage mais le tirage est plus lent. Ces méthodes offrent des alternatives intéressantes à l'utilisation de marches aléatoires isotropes pour explorer des modèles, que ce soit à des fins de simulation, de test, ou de model-checking.

Le choix de la borne n sur la longueur des chemins peut sembler difficile à définir, mais en fonction du critère de couverture désiré, il est souvent facile à déterminer. Par exemple, si le critère est de considérer les chemins qui passent au plus une fois dans chaque boucle, n est égal à la longueur du plus long chemin élémentaire. Dans le domaine du model-checking, cette borne dépend de la propriété à vérifier, mais c'est souvent le diamètre qui est utilisé. Néanmoins, pour les cas où on ne saurait connaître cette borne ou qu'elle soit trop grande, on peut tout à fait choisir arbitrairement un entier n , souvent à partir du nombre d'états du système. Le choix de cet entier présente des analogies avec le choix d'une fonction objectif, comme c'est le cas dans toutes les métaheuristiques qui sont les alternatives actuelles aux marches aléatoires isotropes. Une nouvelle perspective qui permettrait de s'affranchir du choix de cette borne serait d'utiliser le générateur de Boltzmann (Flajolet *et al.*, 2007) dont le paramètre permet de régler la longueur moyenne des chemins obtenus, au lieu d'avoir une longueur fixe.

Remerciements

Je remercie Matthias Krieger pour son aide précieuse qui a permis de trouver une solution au problème d'algèbre linéaire défini à l'équation [5], ainsi qu'Alain Denise et Marie-Claude Gaudel pour leurs conseils avisés.

7. Bibliographie

- Denise A., Gaudel M.-C., Gouraud S.-D., « A Generic Method for Statistical Testing », *ISSRE*, p. 25-34, 2004.
- Denise A., Zimmermann P., « Uniform Random Generation of Decomposable Structures Using Floating-Point Arithmetic », *TCS*, vol. 218, p. 233-248, 1999.
- Dwyer M., Elbaum S., Person S., Purandare R., « Parallel Randomized State-Space Search », *ICSE*, p. 3-12, 2007.
- Enge A., Lefèvre V., Théveny P., Zimmermann P., « Ecole d'été CNC'2 (Calcul Numérique Certifié) », 2009, <http://www.loria.fr/~zimmerma/cnc2.html>.
- Flajolet P., Fusy É., Pivoteau C., « Boltzmann Sampling of Unlabelled Structures », *ANALCO*, vol. 126, SIAM Press, p. 201-211, 2007.
- Flajolet P., Zimmermann P., Cutsem B. V., « A Calculus for the Random Generation of Labelled Combinatorial Structures », *TCS*, vol. 132, p. 1-35, 1994.
- Fousse L., Hanrot G., Lefèvre V., Pélissier P., Zimmermann P., « MPFR : A Multiple-Precision Binary Floating-Point Library with Correct Rounding », *ACM Transactions on Mathematical Software*, vol. 33, n° 2, p. 13 :1-13 :15, June, 2007.

- Garavel H., Descoubes N., « Very Large Transition Systems », , <http://tinyurl.com/yuroxx>, July, 2003.
- Garavel H., Lang F., Mateescu R., An overview of CADP 2001, Technical Report n° 254, INRIA, 2001.
- Gaudel M.-C., Denise A., Gouraud S.-D., Lassaïgne R., Oudinet J., Peyronnet S., « Coverage-biased random exploration of large models », *MBT*, 2008.
- Gouraud S.-D., Denise A., Gaudel M.-C., Marre B., « A New Way of Automating Statistical Testing Methods », *ASE*, p. 5-12, 2001.
- Granlund T., « GNU MP : The GNU Multiple Precision Arithmetic Library, version 4.2.4 », , <http://gmpilib.org/manual/>, 2007.
- Grosu R., Smolka S., « Monte Carlo Model Checking », *TACAS*, p. 271-286, 2005.
- Héroult T., Lassaïgne R., Magniette F., Peyronnet S., « Approximate Probabilistic Model Checking », *VMCAI*, p. 73-84, 2004.
- Knuth D. E., *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*, third edn, Addison-Wesley, Boston, MA, USA, 1997.
- Maurer J., Abrahams D., Dawes B., Rivera R., « Boost Random Number Library », , <http://www.boost.org/libs/random/>, June, 2000.
- Siek J., Lee L.-Q., Lumsdaine A., « Boost Random Number Library », , <http://www.boost.org/libs/graph/>, June, 2000.
- Stein W., *SAGE Mathematics Software*, The SAGE Group. 2007, <http://www.sagemath.org>.