

Génération aléatoire de séquence génomiques par la méthode de Boltzmann : Ajout d'un générateur à GenRGenS

Alain Denise - Yann Ponty

16 décembre 2004

1 Génération aléatoire par rejet et méthode de Boltzmann[2]

Une approche pour la génération aléatoire uniforme de séquences de taille n d'un ensemble A_n consiste à procéder *par rejet*. On tire parti du fait qu'il existe parfois des algorithmes de génération aléatoire de complexités efficaces pour un ensemble $B_n \supset A_n$. On engendre alors des séquences de B_n , puis on *rejete* celles qui ne sont pas dans A_n . Si le rapport des cardinaux de $\frac{B_n}{A_n}$ "n'explose" pas trop quand n augmente, alors on tient un algorithme de génération aléatoire polynomial pour la génération uniforme, et même parfois linéaire.

Dans la méthode par rejet, on a relâché la contrainte d'appartenance à l'ensemble considéré, se contentant d'imposer l'appartenance à un surensemble. La méthode dite de *Boltzmann* relâche quant à elle la contrainte de taille. On utilise pour cela un paramètre x qui, en conjonction avec des règles de génération, définit une distribution sur les tailles des séquences engendrées. Quand cette distribution est suffisamment centrée (gaussienne par exemple), ou juste très croissante, il est possible de trouver une relation entre x et la position du *pic de probabilité*. On peut alors positionner ce pic de façon à obtenir une génération par rejet polynomiale, voire linéaire pour une très large classe d'objets combinatoires. L'intérêt de cette méthode est qu'il ne nécessite pas de codage, à partir du moment où la relation entre x et n a été trouvée, ce qui peut être réalisé grâce à un logiciel de calcul scientifique (Maple, Mathematica ...).

2 GenRGenS[1]

GenRGenS est un outil dédié à la génération aléatoire de séquences génomiques. Il intègre d'ores et déjà des algorithmes de génération aléatoire pour les modèles markoviens, les grammaires non contextuelles, les expressions régulières et les modèles hiérarchiques.

Codé en **Java**, il est disponible au format *jar* exécutable ou sous différents formats de compression à l'adresse :

<http://www.lri.fr/~denise/GenRGenS>

Son utilisation permet, entre autre :

- *Une analyses exploratoire des modèles :*
En observant des séquences engendrées, on met en évidence les manquements d'un modèle proposé pour des séquences ou structures génomiques.
- *Un test de pertinence face à la surreprésentation d'un motif :*
Face à l'apparition suspecte d'un motif dans une séquence obéissant à un modèle fortement structuré, on se demande dans quelle mesure cette apparition est un *bruit de fond* lié au modèle ou si elle est la signature d'une fonction. Pour cela, on peut analyser le modèle statistiquement quand une analyse purement mathématique est possible, ou bien engendrer puis analyser des séquences aléatoires issues du modèle.

3 But du Stage

Le but de ce stage est de coder un *addon* pour **GenRGenS** prenant en compte la génération aléatoire selon la méthode de *Boltzmann*. Il prendrait en entrée une description des objets combinatoires à engendrer ainsi que la fonction donnant le paramètre x pour une valeur souhaitée de n_i et engendrerait des séquences avec éventuellement une tolérance ε sur la taille des objets en sortie. La description du modèle serait lue dans un fichier organisé selon un format à définir avec le stagiaire. On pourrait s'inspirer du format interne de *Maple* pour le format des expressions arithmétiques ainsi que des bibliothèques *CombStruct* et *AlgoLib* (INRIA : <http://pauillac.inria.fr/algo/libraries/>) pour la description des objets combinatoires.

4 Outils et langages utilisés au cours du stage

- Java
- JavaCup et JFlex (Yacc et Lex pour Java)
- Maple
- GNUMake,...

Références

- [1] A. Denise, Y. Ponty, and M. Termier. Random generation of structured genomic sequences. In *Proceedings of RECOMB 03*, 2003. Poster.
- [2] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability, and Computing*, 13(4-5):577–625, 2004.