Comment rater la validation de votre algorithme d'ordonnancement

Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, Frédéric Wagner

RenPar'19 / SympA'13 / CFSE'7
Toulouse, France, du 9 au 11 septembre 2009

Comment rater la validation de votre algorithme d'ordonnancement

Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, Frédéric Wagner

Equipes INRIA MOAIS et MESCAL, Laboratoire CNRS LIG, Universités de Grenoble

1. Introduction

Imaginons que vous veniez de développer un nouvel algorithme d'ordonnancement : félicitations ! Pour disposer d'informations qualitatives sur votre algorithme et le comparer à d'autres vous avez décidé comme beaucoup avant vous de réaliser des simulations. Très classiquement vos simulations portent sur des jeux de données aléatoires (ici, des graphes orientés acycliques). Il vous faut désormais choisir une ou plusieurs méthodes de génération pour ces données.
Plutôt que choisir des outils déjà existants [5] parfois même provenant d'autres domaines [1], vous décidez de recoder une méthode sortie de votre imagination. Pas besoin d'aller plus loin, vous venez de rater votre validation : sans analyse fine des caractéristiques des graphes que génère votre méthode vous avez toutes les chances de trébucher sur un biais caché de cette dernière.
Si par un malheureux hasard vous choisiriez d'utiliser des méthodes classiques du domaine de l'ordonnancement, tout espoir n'est pas perdu. Tout d'abord, les méthodes à votre disposition ne sont pas toutes implémentées de façon standardisée. Vous avez donc la possibilité de tomber dans les pièges classiques de l'implémentation d'algorithmes à base de nombre aléatoires (mauvais générateur, mauvaise graine, ...). Dans le cas où les méthodes que vous choisissez sont déjà implémentées, il vous reste deux solutions. Premièrement certaines implémentations disposent d'options en contradiction avec la classe de graphes générés théoriquement par une méthode. C'est le cas par exemple de [4] pour la méthode de Erdős-Rényi présentée plus loin. Deuxièmement, et nous allons nous attarder sur ce point dans la suite, en choisissant correctement une méthode de génération pour comparer votre algorithme aux autres vous pouvez avantager ce dernier. Il suffit pour cela de choisir une méthode générant des graphes aux caractéristiques favorables à votre algorithme mais pas à vos concurrents.

2. Comment deux méthodes peuvent donner des résultats différents

Pour mieux illustrer notre propos nous avons choisi de vous montrer comment deux méthodes bien différentes pour générer des DAGs peuvent modifier radicalement une caractéristique essentielle des graphes en ordonnancement : le plus long chemin.
La première méthode s'appuie sur les modèles de graphes aléatoires de Erdős-Rényi et la notion de matrice d'adjacence d'un graphe. La deuxième nous vient de Winkler [6] et est basée sur les ordres (appelée ici RandomOrders). Il faut en effet rappeler qu'un DAG est un ordre partiel et qu'un ordre partiel est l'intersection de plusieurs ordres totaux.

Algorithme 1 $G(n, p)$: méthode de Erdős-Rényi

ENTRÉES: $n \in \mathbb{N}, p \in \mathbb{R}$.
SORTIES: un graphe avec n nœuds.

```
Initialiser M, matrice d'adjacence nxn à 0
pour tout i de 1 à n faire
  pour tout j de 1 à i faire
    si Random() < p alors
      M[i][j] = 1
    sinon
      M[i][j] = 0
Transformer M en graphe.
```

Algorithme 2 RandomOrders : génération d'un DAG à partir d'un ordre partiel

ENTRÉES: $n, k \in \mathbb{N}$.
SORTIES: un graphe avec n nœuds issu d'un ordre de dimension au plus k.

Générer k ordres totaux (permutation aléatoire des nœuds).
Obtenir un ordre partiel par intersection de ces k ordres.
Transformer l'ordre partiel obtenu en DAG.

Le plus long chemin (aussi appelé *chemin critique*) est une caractéristique très étudiée dans le domaine

de l'ordonnancement et de nombreux algorithmes y sont sensibles dans leurs performances. Les figures 1 et 2 donnent la distribution du chemin critique pour RandomOrders avec $k = 2$ et $G(n, 0, 5)$ selon différentes valeurs de n.

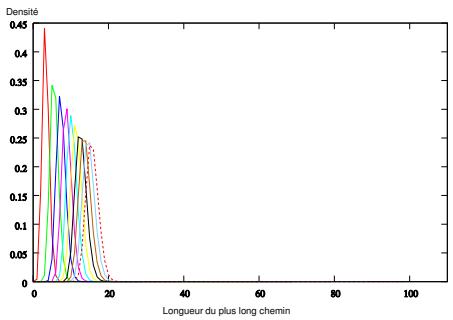

FIG. 1 – Longueur du chemin critique pour Random Orders, $n \in \{10, 20, \ldots, 100\}$ et $k = 2$.

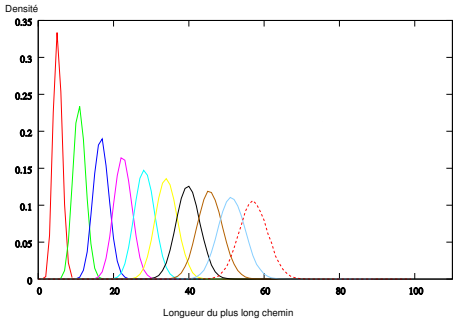

FIG. 2 – Longueur du chemin critique pour $G(n, 0, 5)$, $n \in \{10, 20, \ldots, 100\}$.

Il apparait clairement que suivant la méthode utilisée la longueur moyenne du chemin critique sera variable et donc la performance de l'algorithme étudié modifiée. Il est donc facile de rater votre validation : en fonction des caractèristiques des graphes auquel votre algorithme est sensible choisissez une méthode générant des données vous avantageant.

3. GGen : une boite à outil pour la génération de graphes

Nous venons de vous montrer qu'il peut être très difficile de valider convenablement un algorithme d'ordonnancement. Voilà pourquoi nous avons lancé le projet *GGen*. Il s'agit d'une boite à outils pour la génération de graphes à destination de la validation de politiques d'ordonnancement. Le projet se caractérise surtout par notre volonté d'implémenter les méthodes de générations de graphes connues du domaine de façon normalisée et de fournir systématiquement une analyse poussée des graphes générés. En identifiant clairement quelles sont les difficultés ou les biais que peuvent rencontrer les utilisateurs, nous les aidons à organiser leur campagne de tests et à en tirer des résultats valides.

L'outil manipule le format DOT [2] très connu et dont la simplicité et la diffusion permet de facilement le porter vers n'importe quel format interne pour des simulateurs ou autre.

Étant donnée la difficulté technique de certains aspects de la génération nous nous basons sur des bibliothèques stables et efficaces : la GNU Scientific Library qui fournit une collection de générateurs de nombres aléatoires et de distributions classiques s'appuyant sur des publications et des travaux parfaitement connus et reconnus dans la communauté scientifique [3], ainsi la BOOST Graph Library [4] qui fournit une collection d'algorithmes classiques pour les graphes (plus court chemin, composants connexes, etc.) ainsi que des primitives de gestion des graphes et de leur affichage. GGen est un logiciel libre sous licence CeCILL disponible à l'adresse : `http://ggen.ligforge.imag.fr/`.

Bibliographie

1. G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal Complex Systems*, 1695, 2006.
2. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software : Practice and Experience*, 30(11) :1203–1233, 2000.
3. Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.) : seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
4. Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The boost graph library : user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
5. Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5) :379–394, 2002.
6. Peter Winkler. Random orders. *Order*, 1(4) :317–331, December 1985.