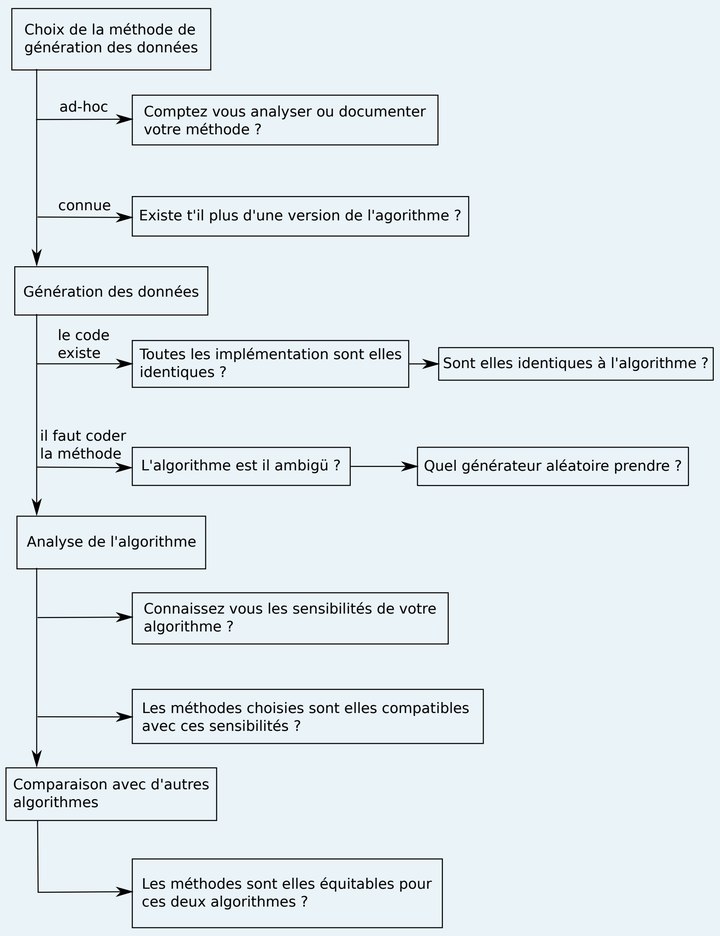
Comment rater la validation de votre algorithme d'ordonnancement

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

Équipes INRIA MOAIS et MESCAL, Laboratoire CNRS LIG, Université de Grenoble

Processus classique : la simulation sur données aléatoires



- Un algorithme d'ordonnancement est sensible à certaines caractéristiques des graphes de taches manipulés.
- Pour comparer correctement deux algorithmes leur simulation doit s'effectuer sur des collections de graphes *équitables*.

Caractéristiques sur les graphes à prendre en compte

- chemin critique (le plus long)
- distribution des degrés
- nombre chromatique
- ensemble indépendant maximal
- nombre d'arêtes
- taille / nombre de composantes connexes
- et bien d'autres

Identification du problème

Générer des graphes pour la comparaison d'algorithmes signifie :

- choisir des méthodes connues,
- spécifier exactement leur interface (entrées, sorties),
- disposer de caractéristiques théoriques sur les graphes générés,
- implémenter ces méthodes,
- vérifier que l'implémentation ne fait pas varier les spécifications techniques et théoriques,
- analyser en profondeur ces méthodes et l'influence des graphes sur les ordonnanceurs étudiés.

Deux méthodes de génération aux caractéristiques différentes

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.

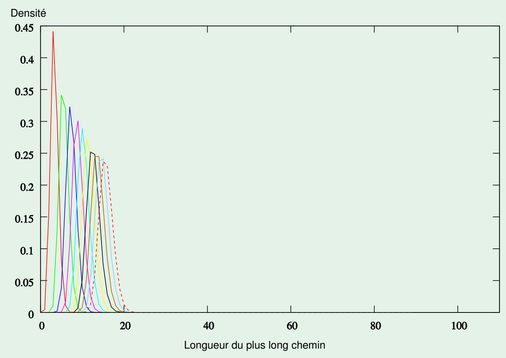


Fig.: Longueur du chemin critique pour Random Orders, $n \in \{10, 20, \ldots, 100\}$ et $k = 2$.

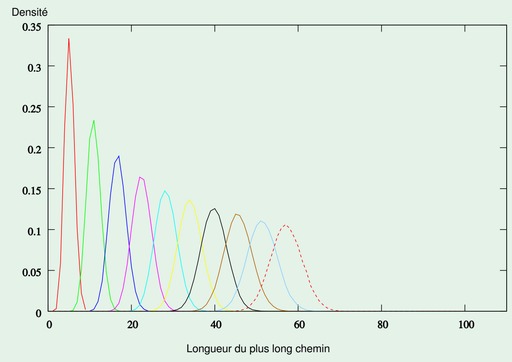


Fig.: Longueur du chemin critique pour $G(n, 0, 5)$, $n \in \{10, 20, \ldots, 100\}$.

Autres sources de biais

- divergence d'interprétation des algorithmes,
- mauvais générateurs aléatoires,
- mauvaise paramétrisation des algorithmes,
- bugs, ...

Il nous manque :

- Une spécification pour chaque méthode de génération de graphes :
  - algorithme clair et précis,
  - spécification théorique des entrées et sorties
  - validation intensive de l'implémentation
- Une grille d'analyse et des outils pour caractériser les graphes générés par chaque méthode.

GGen : une boite à outil pour la génération de graphes

- Coté génération : spécification et implémentation rigoureuse des algorithmes les plus classiques.
- Coté analyse : un large panel d'algorithmes sur les graphes et une grille d'analyse des méthodes.

Conception

- Un format de sortie standard et facile d'accès : DOT.
- Des bibliothèques spécialisées pour le travail délicat : BOOST Graph Library (manipulation de graphes) et GNU Scientific Library (nombres aléatoires).
- Des campagnes régulières de validation.

Travaux encore en cours

- Identifier les points délicats dans l'implémentation de chaque méthode, les documenter et les couvrir par des tests.
- Implémenter des versions erronées des algorithmes pour identifier clairement les problèmes qu'ils posent.
- Lancer une grande campagne de génération de graphes pour étudier sous tous les angles leurs caractéristiques.

Renseignements supplémentaires

- Logiciel sous licence libre CeCiLL.
- Disponible à cette adresse : `http://ggen.ligforge.imag.fr`.
- Ce projet est soutenu par le GDR Recherche Opérationnelle.