# TRAVAIL PRATIQUE 2
# INF4705 – Automne 2016


POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Le 05 novembre 2016

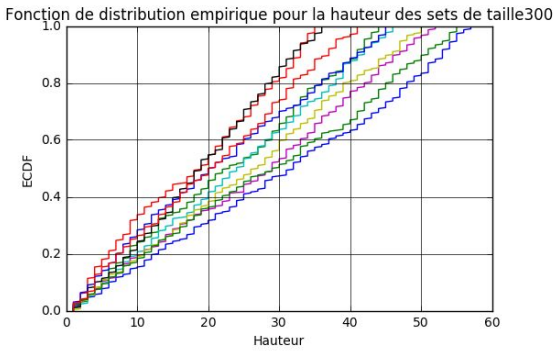
Adrien Doumergue (1868995) & Robin Royer (1860715)

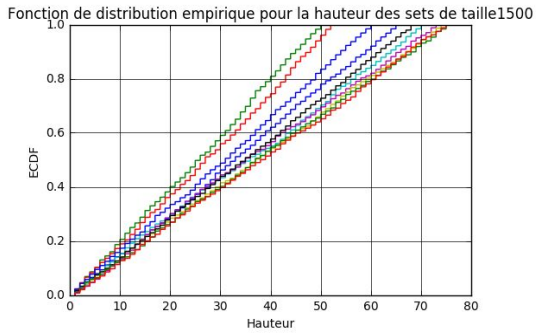
# Introduction

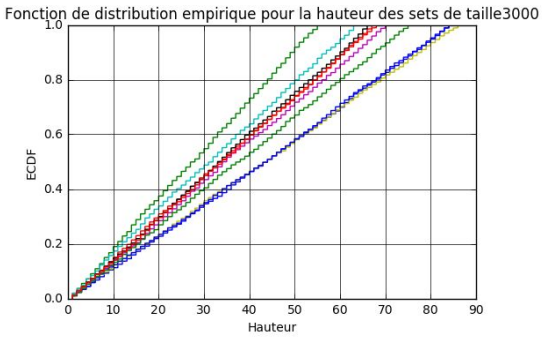
Ce TP propose une approche algorithmique qui permet de déterminer, à partir d'un set de blocs, la tour la plus haute possible. En fonction des exigences de performances, de consommation en temps et en espace mémoire, nous allons étudier trois algorithmes et nous verrons dans quelle situation ils sont le mieux appropriées.

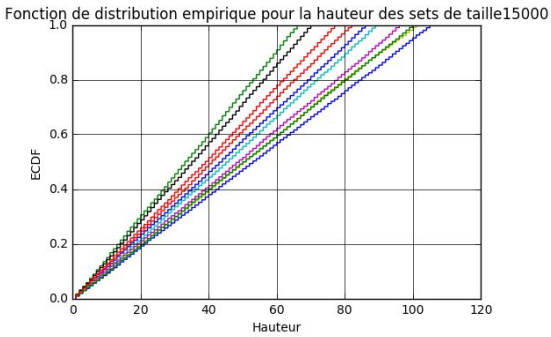
# Analyse des jeux de données:

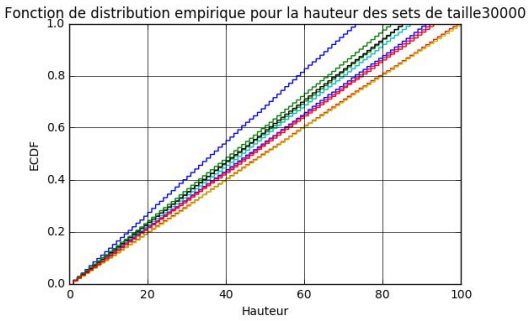
On ne va étudier que les hauteurs pour les sets de blocs contenant toutes les transformations possibles de chaque blocs. Ainsi, on étudiera en réalité l'ensemble des informations possibles pour les sets de blocs de départ.

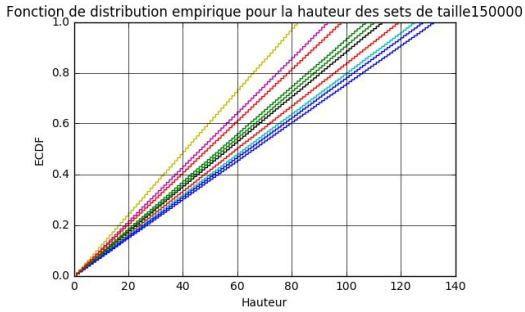


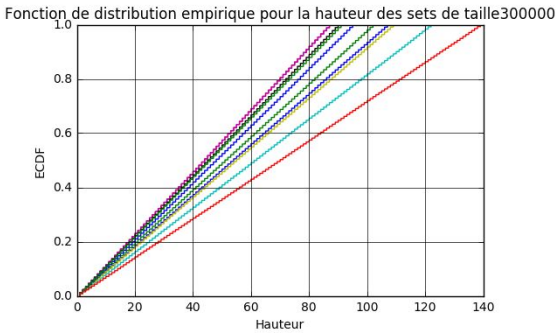



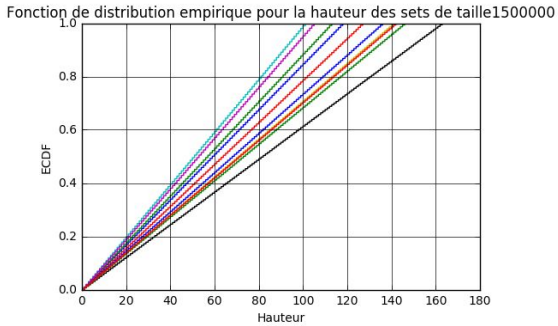



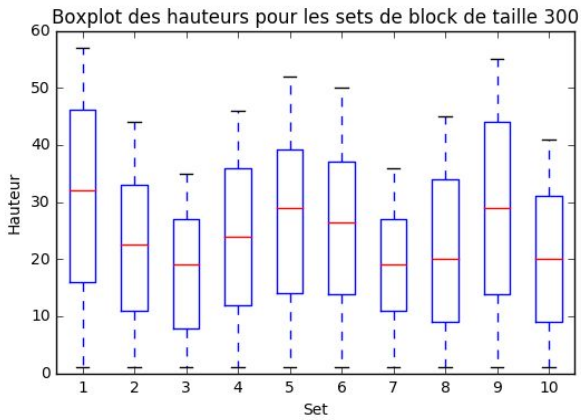


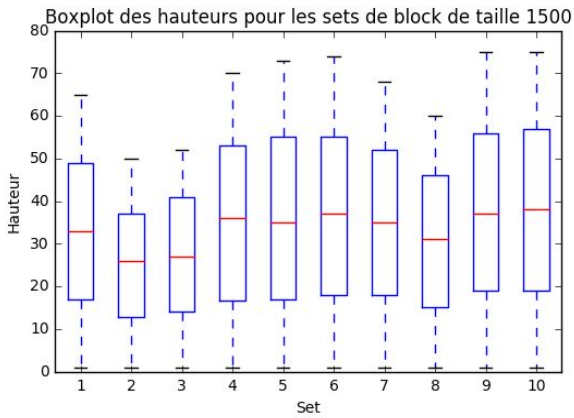



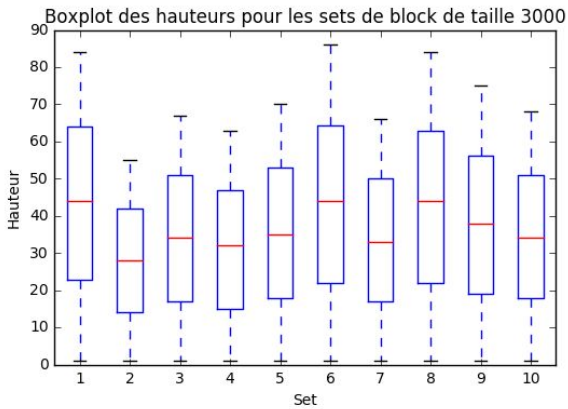



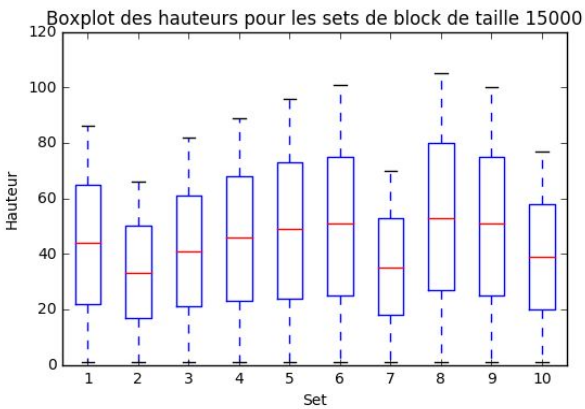



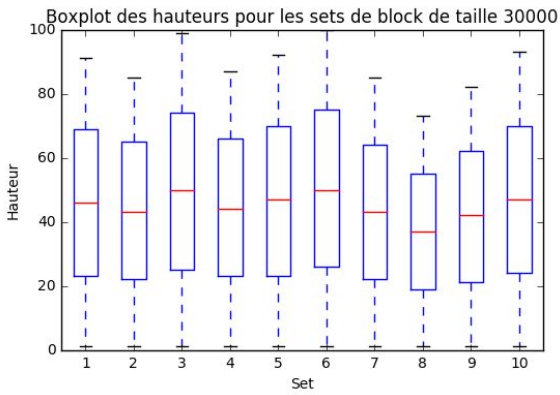

En affichant les fonctions de distribution empirique de chacun des sets de blocs, on se rend compte que chacun des sets a été créé selon une loi de répartition uniforme entre 1 et un max qui dépend de la taille du set mais qui est aléatoire : c'est pourquoi les fonctions de distribution sont linéaires mais n'ont pas la même pente.

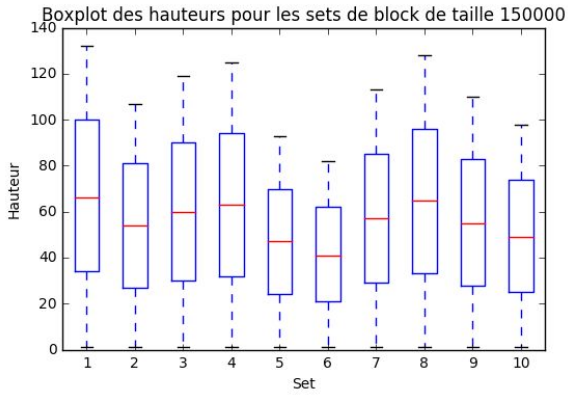


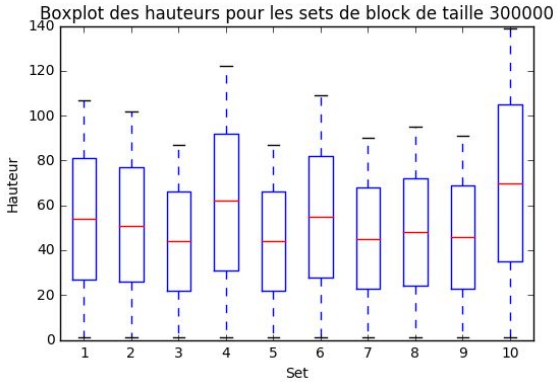



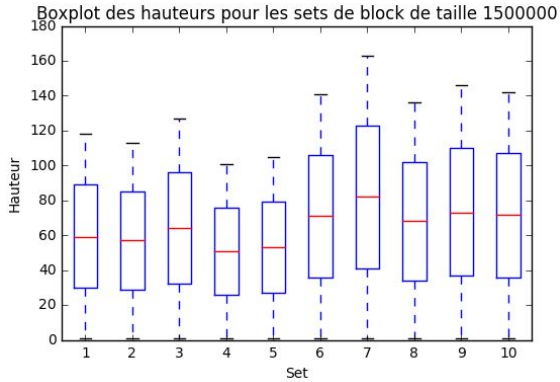






En affichant les boxplot, on voit bien que les “maximums moyens” augmentent avec la taille des sets de blocs.

# Résultats globaux :

**Vorace :** Notre algorithme trie la liste et sélectionne dans la liste triée un bloc qui peut être posé. Il est alors posé sur la tour avec une probabilité qui dépend de la hauteur du blocs sélectionnée.

| **Taille** | **Temps moyen (ms)** | **Hauteur moyenne** |
|---|---|---|
| 100 | 0.58 | 434 |
| 500 | 3.49 | 1219 |
| 1000 | 8.01 | 1683 |
| 5000 | 51.90 | 3366 |
| 10 000 | 132.14 | 4173 |
| 50 000 | 874.54 | 8132 |
| 100 000 | 1885.9 | 8330 |
| 500 000 | 12640 | 15206 |

**Tabou :** cf énoncé

| Taille | Temps moyen (ms) | Hauteur moyenne |
|---|---|---|
| 100 | 9.93 | 523 |
| 500 | 22.20 | 1141 |
| 1000 | 22.59 | 1309 |
| 5000 | 26.85 | 1833 |
| 10 000 | 32.15 | 1919 |
| 50 000 | 38.72 | 2510 |
| 100 000 | 35.72 | 2364 |
| 500 000 | 59.77 | 3758 |

**Dynamique :** Notre algorithme trie selon l'aire les blocs et pour chaque bloc, elle va crée une nouvelle tour qu'elle ajoute dans un set de tours où l'on pose le bloc sélectionné sur la tour de notre ancien set qui l'accepte et maximise la hauteur. S'il n'y a aucune tour qui accepte ce bloc, on crée une tour avec uniquement le bloc sélectionné. Ceci assure de renvoyer la plus haute tour en recherchant à la fin la tour la plus haute parmi notre set de tours.

| Taille | Temps moyen (ms) | Hauteur moyenne |
|---|---|---|
| 100 | 0.0818136215209961 | 614.5 |
| 500 | 6.04888162612915 | 1705.5 |
| 1000 | 43.012985181808475 | 2325.4 |

# Analyse asymptotique

Afin de comparer les 3 algorithmes que nous avons implémentés, nous avons effectué leur analyse asymptotique:

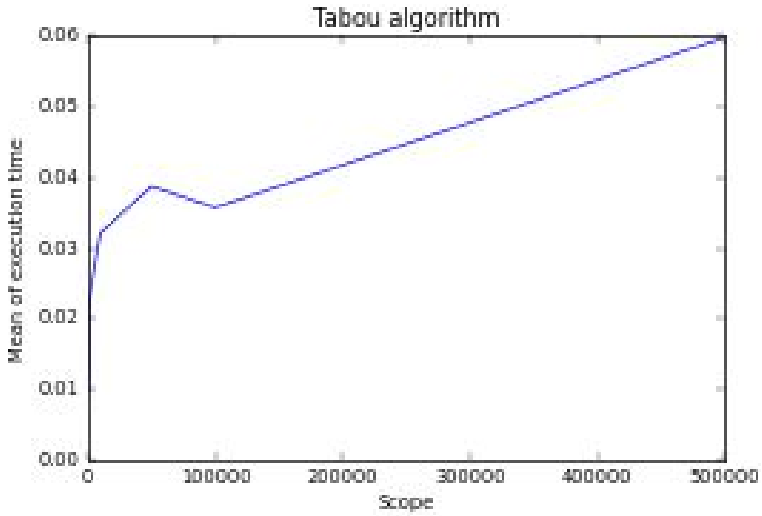
- Vorace $\Theta(n * log(n))$
- Dynamique$\Theta(n^2)$
- Tabou$\Theta(1)$

# Analyse hybride

Nous avons fait le choix d'effectuer un test de rapport pour ces algorithmes.

En effet, nous avons une idee assez precise de la consommation des algorithmes grâce à notre analyse asymptotique, le test des rapports est alors totalement indiqué et nous permet de valider ou d'invalider notre précédente analyse.

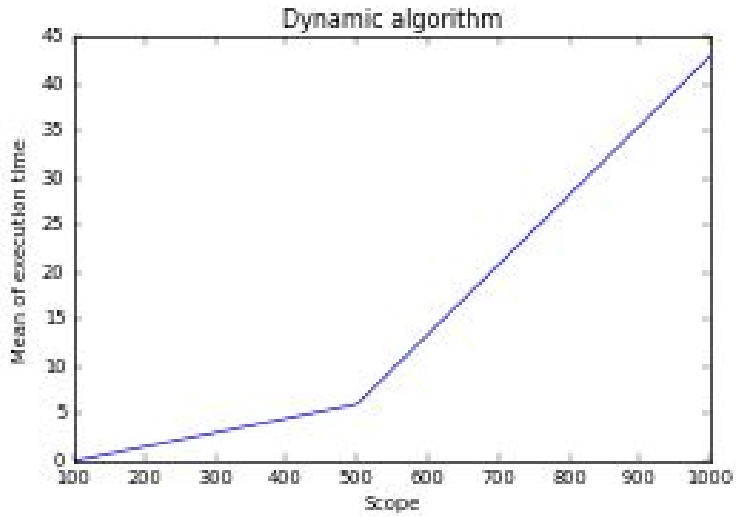
## Tendance générale de l'algorithme tabou




Nous remarquons que le temps d'exécution est compris entre 0.02 et 0.06 l'impact de la taille des exemplaires est donc négligeable et la complexité  est donc constante.

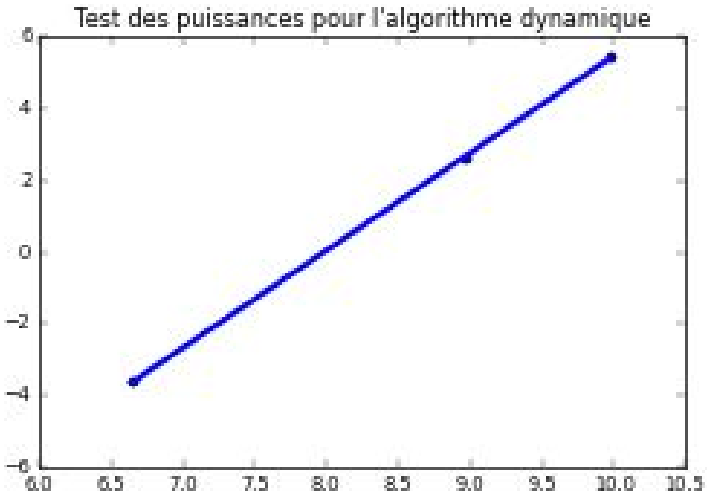
Il devient alors inutile d'effectuer le test de rapport ou des constantes sur cette algorithme (nous n'aurions qu'un point un abscisse.)

## Tendance générale de l'algorithme dynamique



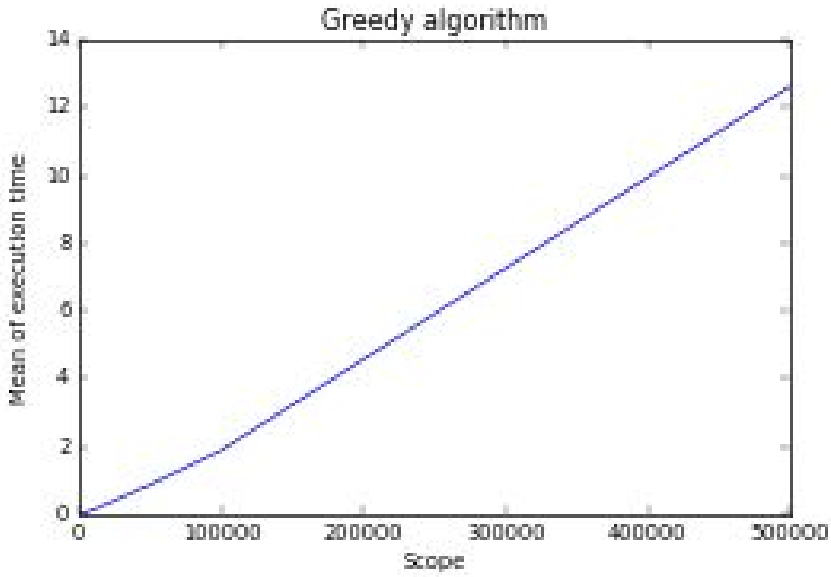
La tendance pour l'algorithme dynamique donne une allure grossière $y = x^2$ .Nous avons peu de valeurs à exploiter car il s'agit d'un algorithme très lent pour des exemplaires de taille élevé et nous n'avons pas pu calculer les moyennes pour des exemplaires de taille 5000 et plus.
Nous n'avons pas effectué de test du rapport car nous avons seulement 3 valeurs et nous n'aurions pas pu déterminer si la courbe convergeait.

Nous avons donc effectué un test de puissance pour confirmer notre analyse asymptotique :



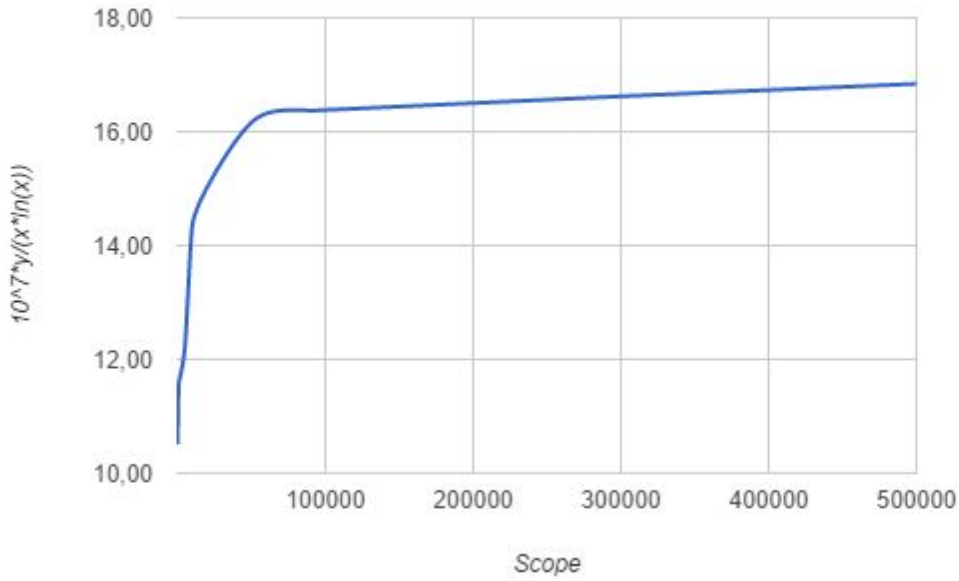

On obtient une pente de 2.7 ce qui nous peut amener à penser que notre analyse est biaisé mais il faut rappeler qu'il s'agit de tailles d'exemplaire faibles qui ne traduisent pas forcément le caractère asymptotique.

## Tendance générale de l'algorithme vorace :




On peut remarquer qu'il suit une loi presque linéaire : en effet, l'algorithme vorace n'effectue qu'un sort du set de blocs et le parcourt : on prévoit donc une complexité $\boldsymbol{\Theta}(n * log(n))$

Test du rapport de l'algorithme vorace




On peut remarquer que la fonction associé au test du rapport converge vers une valeur **b=16,8.10^(-7)** ce qui confirme notre hypothèse selon laquelle l'algorithme vorace évolue en **Θ(** $n * log(n)$ **).** La valeur trouvée est alors notre constante multiplicative.

# Discussion des algorithmes

L'algorithme dynamique renvoie la meilleure tour donc il est le plus performant mais ses capacités sont limités, par sa consommation de ressources très importante (temps de calculs et espace mémoire), au traitement d'exemplaires de taille réduite.

L'algorithme vorace propose une bonne alternative entre performances et consommation car il renvoie une valeur proche de la réalité (ils se trompent d'au plus 30% pour les sets de taille 100, 500 et 1000). Sa consommation est intéressante car elle contient des constantes très petites avec une complexité qui reste correcte : **Θ(** $n * log(n)$ **).**

L'algorithme tabou, quant à lui, renvoie très rapidement et un utilisant très peu de mémoire, un résultat avec peu d'intérêt, indépendamment de la taille de l'exemplaire considéré.
Il peut alors être très intéressant pour des exemplaires de tailles astronomiques et des ressources très limités ou lorsque que l'on veut une solution afin de rapidement itérer sur un résultat.

# Conclusion

Pour obtenir une idée en un temps très court de l'ordre de grandeur de la tour, l'algorithme tabou est une bonne option. Cependant, il ne délivre pas une réponse optimale,  il faudra alors se tourner vers des algorithmes plus gourmand en temps de calcul ou en espace mémoire afin de converger vers une solution optimal (local ou global).

# Source

*Régression linéaire :*
bibliothèque sklearn de Python