

# Evaluation des algorithmes de ranking pour ...

Flavie Bertrand et Marion Tremblay

2025-12-08

## 1. Description du problème et objectif

### 1.1. Problème général de ranking

On considère :

- $i \in \{1, 2, \dots, n\}$  : l'élément  $i$
- un score associé à chaque élément :  $s_i \in \mathbb{R}$
- une variable binaire  $x_{i,p} \in \{0, 1\}$  qui vaut 1 si l'élément  $i$  est placé à la position  $p$  (0 sinon)
- un poids  $w_p$  associé à la position  $p$

L'objectif est de déterminer un classement des  $k$  premiers éléments maximisant le score total.

$$\max_x \sum_{i=1}^n \sum_{p=1}^k w_p s_i x_{i,p}$$

**Sous contraintes :**

- Un élément ne peut occuper au plus qu'une seule position :

$$\sum_{p=1}^k x_{i,p} \leq 1, \quad \forall i$$

- Chaque position doit être occupée par exactement un élément :

$$\sum_{i=1}^n x_{i,p} = 1, \quad \forall p$$

- Contraintes de groupes :

On définit des groupes :

$$G_g \subseteq \{1, \dots, n\}, \quad g = 1, \dots, G$$

On impose que chaque groupe  $G_g$  apparaisse au plus  $\alpha_g$  fois dans les  $k$  premiers :

$$\sum_{i \in G_g} \sum_{p=1}^k x_{i,p} \leq \alpha_g, \quad \forall g$$

- 1.2. Contexte et exemple concret
2. Approche heuristique
3. Solution 1 : programmation dynamique
4. Solution 2 : programmation dynamique et tas
5. Comparaison des résultats
6. Complexité des algorithmes (par le calcul)

Voici nos 4 algorithmes :

1. Algorithme naïf **R**
2. Algorithme dynamique **R**
3. Algorithme dynamique C++ (DP exacte)
4. Algorithme dynamique C++ amélioré (beam search)

Pour chacun, nous analysons :

- La complexité **temps** (meilleur, pire, moyenne)
- La complexité **mémoire**
- Les paramètres en jeu :
  - $n$  : nombre d'items
  - $G$  : nombre de groupes
  - $k$  : cardinal maximal
  - $m$  : nombre de groupes actifs
  - $S$  : nombre d'états atteints en DP
  - $M$  : borne combinatoire théorique des états

### 6.1. Algorithme naïf en **R**

Dans cette algorithme, à chaque étape on explore tous les candidats restants pour vérifier la faisabilité des groupes avant de choisir celui au score maximal.

#### 6.1.1 Complexité théorique

On rappelle :

- Vérification faisabilité groupe = coût  $\Theta(G)$
- Recherche du max =  $\Theta(n)$

**Meilleur cas**

$$T(n) = \Theta(n \log n)$$

(le tri initial domine si la sélection s'arrête tôt ou si  $k = O(1)$ ).

**Pire cas**

$$T(n) = \Theta(Gn^2)$$

(ou  $\Theta(n^2)$  si  $G = O(1)$ ).

**Cas moyen**

$$T(n) = \Theta(Gn^2)$$

**Espace**

$$\Theta(n + m)$$

## 6.2. Algorithme dynamique en R

Cet algorithme construit un tableau de DP associant à chaque état un score maximal.

### 6.2.1 Rappels sur les états

Un état est défini par :

$$\text{état} = (s, c_1, c_2, \dots, c_G)$$

où  $c_g$  est le nombre d'items du groupe  $g$ .

Nombre théorique maximal d'états :

$$M = \prod_{g=1}^G (\max\_cap_g + 1)$$

### 6.2.2 Complexité

**Meilleur cas**

$$T(n) = \Theta(nk)$$

si très peu d'états sont peuplés.

**Pire cas**

$$T(n) = \Theta(nkSm)$$

avec  $S = M$ , donc exponentiel en  $G$ .

**Cas moyen**

$$T(n) = \Theta(nkmS)$$

**Espace**

$$\Theta(nm + nkS)$$

### 6.3. Algorithme dynamique C++ (exact)

Version C++ optimisée utilisant `unordered_map` mais explorant encore **tous les états utiles**.

**Meilleur cas**

$$T(n) = O(nkG)$$

Si  $S = O(1)$ , très petit nombre d'états actifs.

Si en plus  $k = O(1), G = O(1)$  :

$$T(n) = \Theta(n)$$

**Pire cas**

$$T(n) = \Theta(nkMG)$$

où :

$$M = \prod_{g=1}^G (\max\_cap_g + 1)$$

exponentiel en  $G$ .

**Cas moyen**

$$T(n) = O(nkSG)$$

avec  $S \ll M$  en pratique.

**Espace**

Même ordre de grandeur que R :

$$\Theta(kSG)$$

### 6.4. Algorithme dynamique amélioré C++ (Beam Search)

On conserve uniquement les **beam\_size** meilleurs états à chaque niveau de DP.

#### 6.4.1. Complexité

**Meilleur cas** (`beam_size = 1`)

$$T(n) = O(nkG)$$

**Pire cas**

$$T(n) = O(nkS(G + \log S))$$

où

$$S = \min \left( \text{beam\_size}, \prod_{g=1}^G (\text{max\_cap}_g + 1) \right)$$

Cas moyen

$$T(n) = O(nkSG)$$

avec un  $S$  modéré et contrôlé par `beam_size`.

Espace

$$\Theta(kSG)$$

## 6.5. Graphiques comparatifs

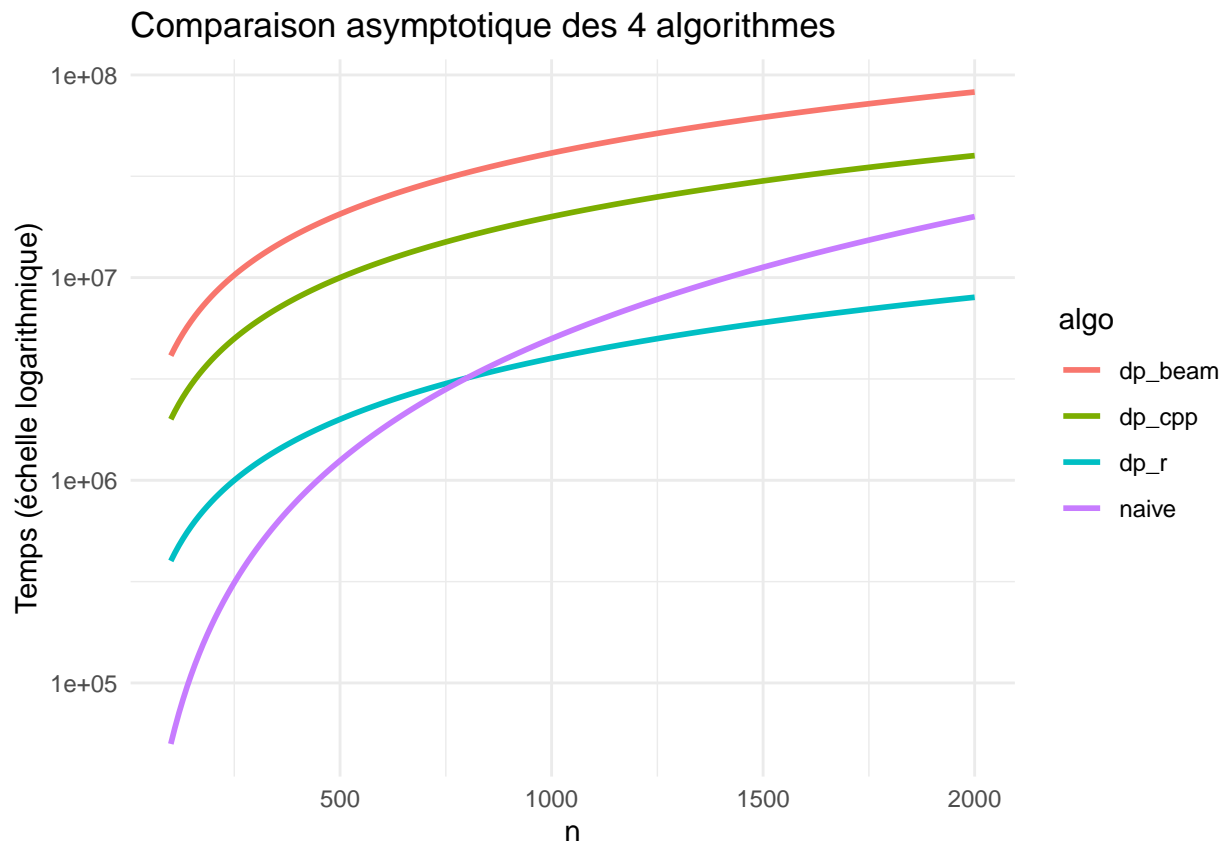
```
library(ggplot2)
library(dplyr)
library(tidyr)

n <- seq(100, 2000, length.out = 200)
G <- 5
k <- 20
S <- 200
M <- (3+1)^G

df <- data.frame(
  n = n,
  naive = n^2 * G,
  dp_r = n * k * S,
  dp_cpp = n * k * S * G,
  dp_beam = n * k * S * (G + log(S))
)

df_long <- df %>%
  pivot_longer(-n, names_to="algo", values_to="T")

ggplot(df_long, aes(n, T, color=algo)) +
  geom_line(size=1) +
  scale_y_log10() +
  theme_minimal() +
  labs(
    title="Comparaison asymptotique des 4 algorithmes",
    y="Temps (échelle logarithmique)",
    x="n"
  )
```



## 7. Temps de calcul