Algorithme de Viterbi — Rapport M2

Mohamed Skander Gharbi

2025-04-11

```
# Installation du package depuis GitHub
 devtools::install_github("Skan1511/M2algorithmique_Viterbi")
```

```
## Rcpp (1.0.13-1 -> 1.0.14) [CRAN]
```

```
## package 'Rcpp' successfully unpacked and MD5 sums checked
```

```
##
## The downloaded binary packages are in
##  C:\Users\gharb\AppData\Local\Temp\RtmpMhB8db\downloaded_packages
## -- R CMD build -------------------------------------------------------------
##          checking for file 'C:\Users\gharb\AppData\Local\Temp\RtmpMhB8db\remotes12f44c6739c5\Skan151
##       -  preparing 'viterbiM2': (886ms)
##    checking DESCRIPTION meta-information ...  v  checking DESCRIPTION meta-information
## -  cleaning src
##       -  checking for LF line-endings in source and make files and shell scripts
##   -  checking for empty or unneeded directories
##   -  looking to see if a 'data/datalist' file should be added
##       -  building 'viterbiM2_0.1.0.tar.gz'
##
##
```

```
# Chargement des packages
library(gtools)
library(ggplot2)
library(reshape2)
library(Biostrings)

# Chargement des fonctions du projet
source("C:/Users/gharb/Documents/M2algo_viterbi/viterbiM2/src/naive_R/viterbi_naive.R")
```

```
## [1] "Mon premier script Viterbi (version naive)"
## [1] "Pluie"  "Soleil" "Pluie"
```

```
source("C:/Users/gharb/Documents/M2algo_viterbi/viterbiM2/src/viterbi_R/viterbi_algo.R")
```

```
## [1] "Pluie"  "Soleil" "Pluie"
```

```
source("C:/Users/gharb/Documents/M2algo_viterbi/viterbiM2/simulations/viterbi_brca1.R")
```

```
## Matrice de transition A :
##             C           N
## C 0.703406814 0.002004008
## N 0.002004008 0.292585170
##
## Matrice d'émission B :
##           A         T         C         G
## C 0.2585227 0.3011364 0.2357955 0.2045455
## N 0.2857143 0.2925170 0.1632653 0.2585034
```

Introduction

Dans ce projet, nous étudions deux approches pour résoudre le problème du décodage dans un modèle de Markov caché (HMM) : - Une **approche naïve** par force brute, - L'**algorithme de Viterbi**, basé sur la programmation dynamique.

Nous appliquons d'abord ces méthodes sur un exemple jouet (Pluie/Soleil), puis sur des **séquences d'ADN simulées** contenant des régions codantes et non codantes.

Exemple jouet : Modèle Pluie/Soleil

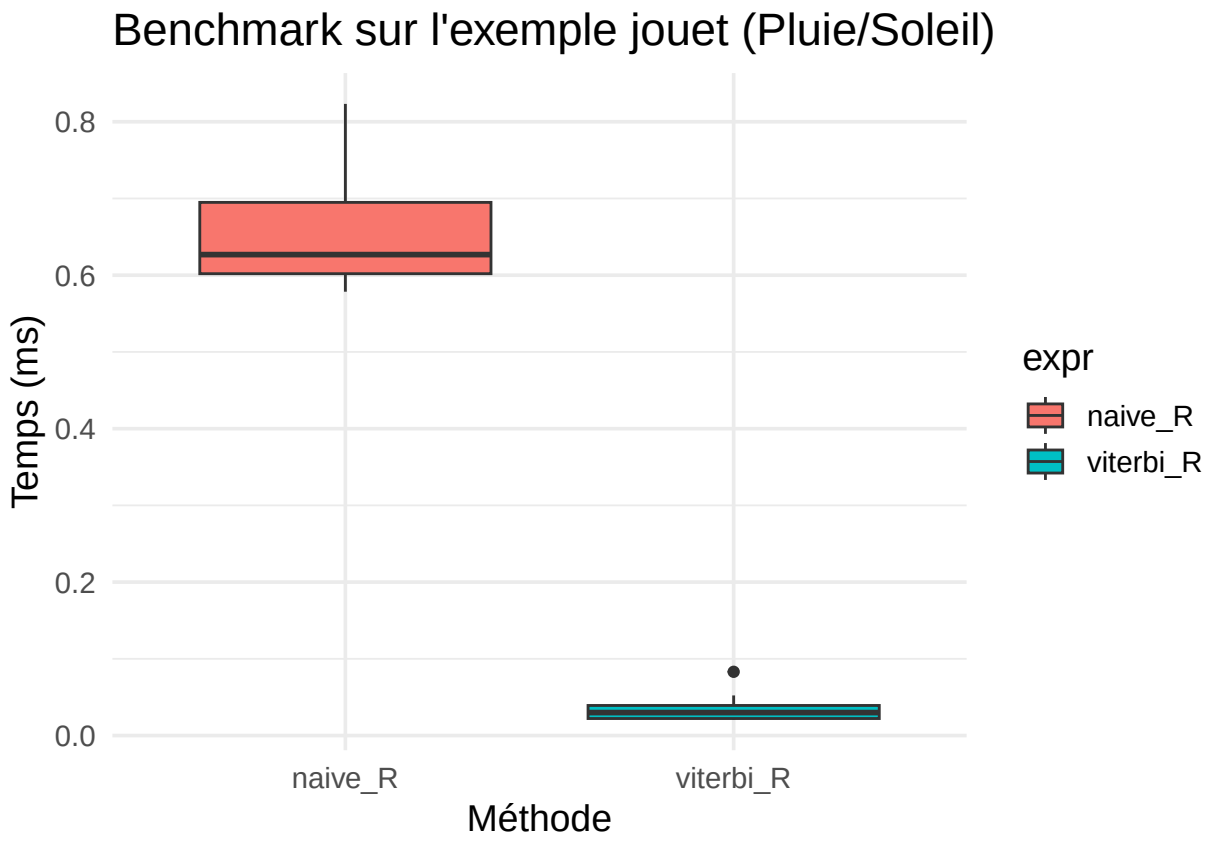
```
states_ps <- c("Pluie", "Soleil")
obs_ps <- c(1, 2, 1, 2, 1)
A_ps <- matrix(c(0.7, 0.3, 0.4, 0.6), nrow = 2, byrow = TRUE)
B_ps <- matrix(c(0.9, 0.1, 0.2, 0.8), nrow = 2, byrow = TRUE)
pi_ps <- c(0.6, 0.4)

res_naive_ps <- trouver_sequence_naive(obs_ps, states_ps, pi_ps, A_ps, B_ps)
res_vit_ps <- viterbi(obs_ps, states_ps, pi_ps, A_ps, B_ps)

cat("Méthode Naïve (Pluie/Soleil):\n"); print(res_naive_ps)
```

```
## Méthode Naïve (Pluie/Soleil):

## [1] "Pluie"  "Soleil" "Pluie"  "Soleil" "Pluie"
```

```
cat("\nMéthode Viterbi (Pluie/Soleil):\n"); print(res_vit_ps)
```

```
##
## Méthode Viterbi (Pluie/Soleil):

## [1] "Pluie"  "Soleil" "Pluie"  "Soleil" "Pluie"
```

```r
library(microbenchmark)

bm_jouet <- microbenchmark(
  naive_R = trouver_sequence_naive(obs_ps, states_ps, pi_ps, A_ps, B_ps),
  viterbi_R = viterbi(obs_ps, states_ps, pi_ps, A_ps, B_ps),
  times = 10L
)

df_jouet <- as.data.frame(bm_jouet)
ggplot(df_jouet, aes(x = expr, y = time / 1e6, fill = expr)) +
  geom_boxplot() +
  labs(
    title = "Benchmark sur l'exemple jouet (Pluie/Soleil)",
    x = "Méthode",
    y = "Temps (ms)"
  ) +
  theme_minimal(base_size = 14)
```



Modèle HMM : Rappel

Un modèle de Markov caché est défini par : - Un ensemble d'états cachés $S = \{s_1, s_2, \ldots, s_N\}$ - Une séquence d'observations $O = \{o_1, o_2, \ldots, o_T\}$ - Une matrice de transition A - Une matrice d'émission B - Un vecteur de probabilités initiales π

Notre objectif est d'estimer la séquence d'états la plus probable ayant généré la suite d'observations.

Simulation d'une séquence ADN

```
set.seed(42)
seq_length <- 500
bases <- c("A", "T", "C", "G")
seq <- paste(sample(bases, seq_length, replace = TRUE), collapse = "")
seq <- DNAString(seq)
states <- rep("N", seq_length)
cds_start <- sample(1:(seq_length - 100), 10, replace = TRUE)
cds_end <- cds_start + 100
for (i in 1:length(cds_start)) {
  states[cds_start[i]:cds_end[i]] <- "C"
}
```

Estimation des matrices de transition et d'émission

```
count_CC <- count_CN <- count_NC <- count_NN <- 0
count_A_C <- count_T_C <- count_C_C <- count_G_C <- 0
count_A_N <- count_T_N <- count_C_N <- count_G_N <- 0

for (i in 2:seq_length) {
  if (states[i-1] == "C" && states[i] == "C") count_CC <- count_CC + 1
  else if (states[i-1] == "C") count_CN <- count_CN + 1
  else if (states[i-1] == "N" && states[i] == "C") count_NC <- count_NC + 1
  else count_NN <- count_NN + 1

  base <- substring(as.character(seq), i, i)
  if (states[i] == "C") {
    if (base == "A") count_A_C <- count_A_C + 1
    if (base == "T") count_T_C <- count_T_C + 1
    if (base == "C") count_C_C <- count_C_C + 1
    if (base == "G") count_G_C <- count_G_C + 1
  } else {
    if (base == "A") count_A_N <- count_A_N + 1
    if (base == "T") count_T_N <- count_T_N + 1
    if (base == "C") count_C_N <- count_C_N + 1
    if (base == "G") count_G_N <- count_G_N + 1
  }
}

A <- matrix(c(count_CC, count_CN, count_NC, count_NN), nrow = 2, byrow = TRUE)
A <- A / rowSums(A)
rownames(A) <- colnames(A) <- c("C", "N")

B <- matrix(c(count_A_C, count_T_C, count_C_C, count_G_C,
              count_A_N, count_T_N, count_C_N, count_G_N),
            nrow = 2, byrow = TRUE)
```

```r
B <- B / rowSums(B)
rownames(B) <- c("C", "N")
colnames(B) <- c("A", "T", "C", "G")

pi <- c(sum(states == "C"), sum(states == "N"))
pi <- pi / sum(pi)
```

Application des algorithmes sur ADN

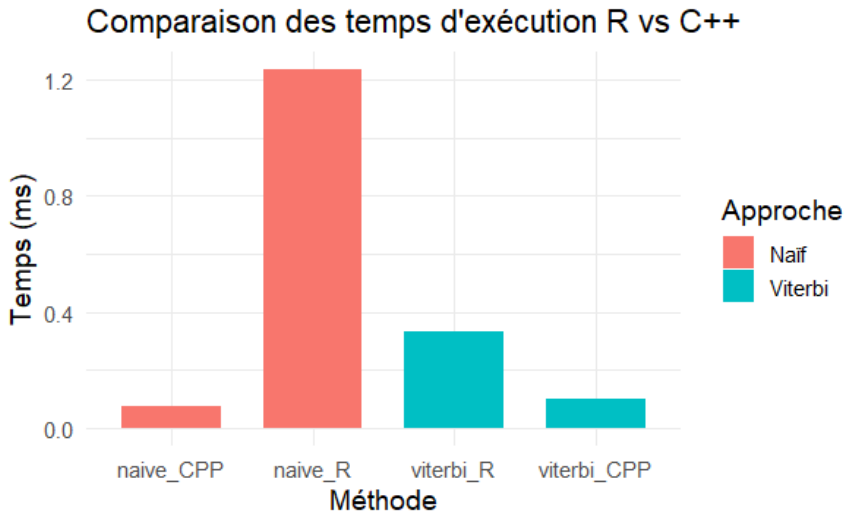
```r
obs_small <- sample(1:4, 6, replace = TRUE)
obs_long <- sample(1:4, 50, replace = TRUE)

result_naif <- trouver_sequence_naive_bio(obs_small, c("C", "N"), pi, A, B)
result_viterbi <- viterbi(obs_long, c("C", "N"), pi, A, B)

cat("Résultat Naïf ADN:\n"); print(result_naif)
```

```
## Résultat Naïf ADN:

## [1] "C" "C" "C" "C" "C" "C"
```

```r
cat("\nRésultat Viterbi ADN:\n"); print(result_viterbi)
```

```
##
## Résultat Viterbi ADN:

##  [1] "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C"
## [20] "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C"
## [39] "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C"
```

Nous avons mesuré le temps d'exécution des algorithmes sur des séquences d'ADN simulées. Le graphique suivant montre la comparaison :



```r
source("C:/Users/gharb/Documents/M2algo_viterbi/viterbiM2/simulations/benchmark_cpp_bio.R")
```

```
## Matrice de transition A :
##             C           N
## C 0.703406814 0.002004008
## N 0.002004008 0.292585170
##
## Matrice d'émission B :
##           A         T         C         G
## C 0.2585227 0.3011364 0.2357955 0.2045455
## N 0.2857143 0.2925170 0.1632653 0.2585034
```

```
## Unit: microseconds
##       expr    min     lq    mean median     uq     max neval cld
##    naive_R 1689.0 1750.4 3102.41 1810.8 2087.0 14389.9    10  a
##  naive_CPP    1.7    6.1  182.47   13.3   17.2  1660.8    10   b
## Unit: microseconds
##         expr   min    lq    mean median    uq    max neval cld
##    viterbi_R 277.2 299.0 1257.60  329.4 374.4 9577.2    10   a
##  viterbi_CPP   2.8   4.4  135.99    7.8  16.5 1226.9    10   a
```

Conclusion

Nous avons comparé une méthode naïve et l'algorithme de Viterbi sur des séquences simulées et une application ADN. Viterbi se démarque par son efficacité et sa précision. Cette expérience ouvre la voie à des applications réelles comme l'annotation automatique de génomes.