



ADVANCED ALGORITHMICS AND PROGRAMMING FOR BIOLOGISTS / MODELS,  
METHODS AND ALGORITHMS FOR BIOINFORMATICS

---

# Alignement multiple de traces d'évènements

---

## Rapport de projet

**Année Universitaire :**

2022/2023

**Auteur :**

Elyna BOUCHEREAU

Florent BOYER

**Professeure :**

Christine SINOQUET

# 1 Introduction

L'objectif de ce programme est d'aligner plusieurs chaînes de caractères que nous appellerons traces entre elles, afin de pouvoir les classifier et de démontrer un potentiel lien entre les différentes séquences. Pour pouvoir répondre à cette problématique nous avons décidé de produire un programme en langage C++ qui nous permet de réaliser et d'automatiser cette étape critique en biologie et de ressortir une classification possible de n'importe quelle séquence. Pour cela nous voulons tester notre programme d'alignement multiple sur un fichier texte contenant des séquences aléatoires dans différents cas de figures.

Nous avons validés notre programme d'alignement multiple à partir d'un fichier contenant 20 jeux de données correspondant à une même complexité de séquence générée aléatoirement, avec un nombre d'éléments par séquence de 30.

## 2 Structure de données utilisées

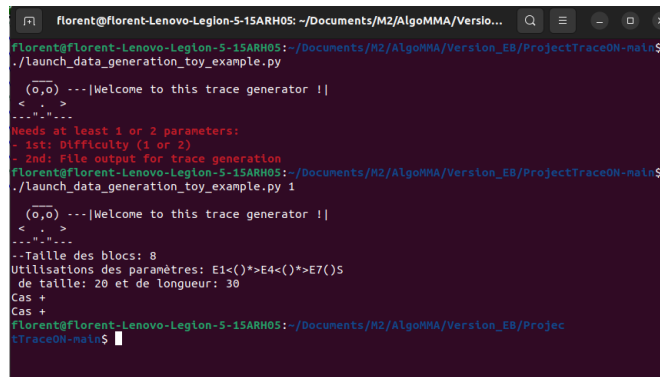
Pour réaliser ce projet nous avons réalisé plusieurs structures de données disponibles dans les fichiers .hpp de notre programme :

- structure : `matri`, `Type_trace`, `cluster_trace`, `tree_ptr`
- classe : `bloc`, `Class_align`, `Multi_Align`, `TArbreBin`

## 3 Génération des données

Chaque jeu de données sera généré aléatoirement à partir d'un fichier spécifiant une expression, le nombre de traces à générer et leur longueur maximum. Pour pouvoir lancer et utiliser ce programme de génération de données nous avons créé un script de lancement en Python, qui prend en compte le choix de difficulté défini par l'utilisateur (1 pour une génération de données dites "facile" et 2 pour une génération de données dites "difficile"). Cette génération de données se situe dans un fichier texte à partir de : "Init.txt" qui contient les paramètres utilisés par le programme pour le niveau facile et difficile, ainsi que le nombre de séquences voulues ainsi que leur taille. Une fois ce fichier lu par le script Python les fichiers `main_TraceON` et `trace_generator` prendront la suite et créeront nos séquences voulues. La génération de données ainsi obtenue va se matérialiser dans un nouveau fichier texte : "Test.txt" qui contient les séquences selon les paramètres précédemment définis.

Pour nos données nous avons utilisé deux types de structures différentes. Une facile :  $E1<()*>E4<()*>E7()S$  et une difficile :  $E2<()+>E8()E12<()E13|E89X3|E14X5>E46<()*>S$ .



```
Florent@Florent-Lenovo-Legion-5-15ARH05: ~/Documents/M2/AlgoMMA/Versio...
Florent@Florent-Lenovo-Legion-5-15ARH05: ~/Documents/M2/AlgoMMA/Version_EB/ProjectTraceON-main$
./launch_data_generation_toy_example.py

(0,0) ---|Welcome to this trace generator !|
< * >
...*...
Needs at least 1 or 2 parameters:
- 1st: Difficulty (1 or 2)
- 2nd: File output for trace generation
Florent@Florent-Lenovo-Legion-5-15ARH05: ~/Documents/M2/AlgoMMA/Version_EB/ProjectTraceON-main$
./launch_data_generation_toy_example.py 1

(0,0) ---|Welcome to this trace generator !|
< * >
...*...
--Taille des blocs: 8
Utilisations des paramètres: E1<()*>E4<()*>E7()S
de taille: 20 et de longueur: 30
Cas +
Cas +
Florent@Florent-Lenovo-Legion-5-15ARH05: ~/Documents/M2/AlgoMMA/Version_EB/Projec
tTraceON-main$
```

FIGURE 1 – Affichage du menu du script Python qui demande à l'utilisateur quel niveau de difficulté utilisé pour son jeu de données.

```

1 #####
2 ##### Init File
3 #####
4 1) Facile:
5 E1<()*>E4<()*>E7(S
6 2) Difficile:
7 E2<()*>E8(E12<())E13|E89X3|E14X5>E46<()*>S
8
9 Nombre_traces:
10 20
11 Taille:
12 30

```

(a) Capture d'écran du fichier "Init.txt" utilisé

```

1 Trace_1| E1 E2 E1 - - - E2 E1 E4 E4 E8 E7 E4 - E6 - - - E7 - - - - - S
2 Trace_2| E1 E1 E2 E2 - - - E1 E1 E4 E6 - E6 E5 - - - E8 - E7 - - - - - S
3 Trace_3| E1 E1 - - - E1 E2 - - - E4 E6 E8 E4 E7 E6 E7 - E8 E7 - - - - - S
4 Trace_4| E1 - - - - - E4 - - - - - E5 - - - E7 - - - - - S
5 Trace_5| E1 - - - - - E1 E1 - - - E4 - - - - - E5 E7 - - - - - S
6 Trace_6| E1 - - - - - E2 - - - - - E4 E5 E7 E8 E7 E6 E4 E4 - E7 - - - - - S
7 Trace_7| E1 - E2 - - - - - E4 E7 E5 E7 E8 E5 E5 E7 E4 E7 - - - - - S
8 Trace_8| E1 - E1 E1 - E2 E1 E1 - E4 - - - - - E8 - E7 - - - - - S
9 Trace_9| E1 E2 E1 E2 E1 E1 E1 E1 E2 E4 E4 - - - E5 E5 - E8 - E7 - - - - - S
10 Trace_10| E1 E2 E2 - - - E2 E2 E2 E4 E7 E5 E7 E6 E7 E8 E7 E4 E7 - - - - - S
11 Trace_11| E1 E2 E1 E1 E2 E1 E2 E1 E1 E4 - E5 - - - E4 - - - E5 E7 - - - - - S
12 Trace_12| E1 E1 - - - E2 E2 E1 E2 E1 E4 - E7 E7 E4 E6 E8 - - - E7 - - - - - S
13 Trace_13| E1 E2 - E1 E2 E2 - E1 - E4 - E5 - - - - - E7 - - - - - S
14 Trace_14| E1 - - - - - E2 - E4 - E5 E5 - - - E8 - - - E7 - - - - - S
15 Trace_15| E1 - - - - - E4 - - - - - E6 - - - E7 - - - - - S
16 Trace_16| E1 E2 E1 E2 E2 E1 - E1 - E4 E5 E6 E5 E4 E5 E8 E8 E7 - - - - - S
17 Trace_17| E1 - - - - - E4 E5 E6 E7 - - - - - E8 E7 - - - - - S
18 Trace_18| E1 E1 - E1 - - - E4 - E8 - - - - - E7 - - - - - S
19 Trace_19| E1 - E2 - E2 E1 - E1 E1 E4 - E8 E6 E5 E4 E6 E6 E7 - - - - - S
20 Trace_20| E1 E2 E2 E1 E1 E2 E1 E2 E1 E4 - - - - - E6 E8 E6 E4 E7 - - - - - S

```

(b) Capture d'écran du fichier "Test.txt" obtenu

FIGURE 2

Une fois ces séquences obtenus nous pouvons continuer et appliquer notre programme d'alignement multiples sur celles-ci.

## 4 Alignement

Pour pouvoir implémenter notre alignement de nos séquences ainsi obtenues, nous avons utilisé la programmation orientée objet possible avec le langage C++ en utilisant des classes avec templates et de la surdéfinition pour faciliter les appels de fonction ou de procédure ainsi qu'éviter la redondance. Nous avons aussi utilisé beaucoup de vecteurs par souci d'efficacité et de clarté du code source. Le principe d'alignement de notre programme suit une approche progressif, plutôt qu'itérative, qui se conforme selon le principe d'une méthode heuristique selon l'algorithme de Needleman et Wunsch. Le fichier "init\_MSA" est le fichier d'initialisation de l'alignement qui contient les pénalités du score d'alignement dû aux gaps (avec 10 pour le premier gap et 1 pour les suivants).

### 4.1 Alignement multiple

Nous commençons par aligner globalement toutes les paires de traces pour construire un arbre binaire des relations évolutives entre les séquences via une matrice de dissimilarité (ou de distance). Les noeuds entre les branches représentent les alignements deux à deux et la racine représente l'alignement complet. Une fois cet arbre construit, le programme prend les deux séquences les plus proches et les fusionne. Puis il crée un nouvel arbre qui progresse vers les séquences plus distantes, remontant ainsi l'arbre initial. Ce programme est rapide pour un nombre raisonnable de séquences longues et plus lent si on aligne un grand nombre de séquences courtes. Les séquences ainsi alignées se trouveront dans le fichier "Test\_output.txt" et les scores associés dans le fichier "Test\_Score.csv".

### 4.2 Matrice de dissimilarité

On effectue un alignement global de toutes les paires possibles dans une matrice triangulaire sans la diagonale, et à partir de cet alignement global on calcule un score de distance entre ces séquences, où plus deux séquences sont proches et plus le score est élevé.

### 4.3 Création d'un arbre binaire

On peut calculer un arbre binaire à partir d'une matrice de dissimilarité par regroupement hiérarchique, dans notre cas nous utiliserons la classification par ascendance hiérarchique (CAH). Le nom des traces apparaît sur nos feuilles et le score obtenu précédemment sur nos noeuds. On commence par regrouper les deux séquences les plus proches, on les fusionne et on réitère sur notre nouvel arbre. Cet arbre sera ensuite utilisé comme guide pour déterminer l'ordre d'incorporation des séquences dans l'alignement multiple. Il sert à identifier les similarités les plus fortes entre les séquences. Les séquences en sortie sont alignées en fonction de l'arbre.

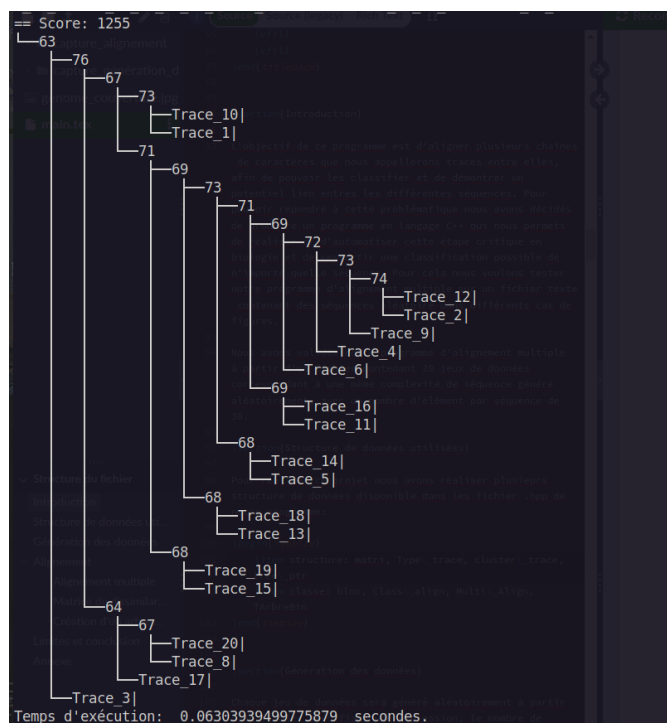


FIGURE 3 – Exemple d'un début d'arbre obtenu après l'alignement multiple.

## 5 Limites et conclusion

Ce programme est rapide pour un nombre raisonnable de séquences longues et plus lent si on aligne un grand nombre de séquences courtes (146" pour 200 traces de 50 de longueur). En effet nous avons éprouver notre programme en l'évaluant sur un nombre conséquent de séquences (2000 traces) et celui-ci casse après avoir rempli la RAM. La génération de beaucoup de traces (500 de 50 de long 0.05") Par contre la longueur limite n'est pas clairement définie, la génération de traces de plus de 100 éléments de long a un coût exponentiel (à 100, 0.30" et à 120 6"). Il serait intéressant d'implémenter à notre programme une méthode différente que celle utilisé pour essayer de palier ce genre de problème.

De plus, il faudrait générer une fonction aléatoire qui génère directement toutes nos génération de données dont on peut avoir besoin. Il aurait fallu aussi continuer à travailler sur la fonction qui doit générer les données dans le cas des "%" car celle-ci ne fonctionne pas correctement, mais nous avons voulu nous concentrer sur l'efficacité et l'optimisation de la méthode d'alignement plutôt que sur la génération de données car nous avons déjà deux niveaux de difficultés de génération de données.

En outre, notre menu qui sert d'interface entre le programme et l'utilisateur est pour le moment très basique et mérite qu'on y passe plus de temps pour aider à l'utilisation de notre programme. Il nous manques aussi une sortie de l'arbre guide obtenus après alignement dans un fichier texte ainsi qu'un fichier de score pour chaque alignement.

Mais aussi, notre plus gros problème dans l'implémentation de notre programme à été lors de l'alignement des séquences avec notamment un soucis sur leur longueur, ainsi que la dernière séquences (peut importe leurs nombres) qui saute pendant l'alignement avec sûrement une erreur lors de l'allocation mémoire mais ils nous à été impossible de déterminer exactement où. Avant de mettre en routine notre code ce problème doit être impérativement résolu. Ce contretemps nous à empêcher aussi de créer le score `proj_length` pour la longueur commune des projections constituant l'alignement multiple. Ainsi que des scores telle que `score_e` qui calculerait le nombre d'événements et `match_e` pour toutes les paires de projections qui eux aussi doivent être implémentés.

Néanmoins nous avons pu prouver que notre alignement multiple était correct grâce au `score_nw` qui correspond aux score de Needleman et Wunsch et au `score_g` qui correspond au comptage du nombre de gap en fonction du nombre de traces alignées.

En conclusion, notre programme n'est pas encore parfait et doit encore être travaillé, mais il réussit son objectif en réalisant un jeu de données aléatoire et à aligner ces séquences de façon robuste.

## 6 Annexe

```
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/M...
(o,o) ---|Welcome to this trace generator !!
< . >
...".-...
20 20

Arguments: 20/20

--Taille bloc5
Utilisations des paramètres: E1<()*>E4<()*>E7()S
de taille: 20 et de longueur: 20
Cas +
Cas +
Temps d'exécution: 0.011456093999981931 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$ ./launch_MSA_test.py

(o,o) ---|Welcome to this program for Multiple Trace Alignment !!
< . /
...".-...
=====
MULTI ALIGN
- Number of sequences: 20
=====
Temps d'exécution: 0.15032659000007698 secondes.
```

(a) Exemple de résultat en exécutant les deux fichiers Python avec un nombre de séquence de 20 de taille 20.

```
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/M...
...".-...
90 20

Arguments: 90/20

--Taille bloc28
Utilisations des paramètres: E1<()*>E4<()*>E7()S
de taille: 20 et de longueur: 90
Cas +
Cas +
Temps d'exécution: 3.016072869000508 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$ ./launch_MSA_test.py

(o,o) ---|Welcome to this program for Multiple Trace Alignment !!
< . /
...".-...
=====
MULTI ALIGN
- Number of sequences: 20
=====
Temps d'exécution: 0.22322910000002594 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$
```

(b) Exemple de résultat en exécutant les deux fichiers Python avec un nombre de séquence de 90 de taille 20.

FIGURE 4

```
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/M...
...".-...
20 1000

Arguments: 20/1000

--Taille bloc5
Utilisations des paramètres: E1<()*>E4<()*>E7()S
de taille: 1000 et de longueur: 20
Cas +
Cas +
Temps d'exécution: 0.10227818200019101 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$ ./launch_MSA_test.py

(o,o) ---|Welcome to this program for Multiple Trace Alignment !!
< . /
...".-...
=====
MULTI ALIGN
- Number of sequences: 1000
=====
Temps d'exécution: 9.337468959000034 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$
```

(a) Exemple de résultat en exécutant les deux fichiers Python avec un nombre de séquence de 20 de taille 1000.

```
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/M...
...".-...
50 500

Arguments: 50/500

--Taille bloc15
Utilisations des paramètres: E1<()*>E4<()*>E7()S
de taille: 500 et de longueur: 50
Cas +
Cas +
Temps d'exécution: 0.10741988699919602 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$ ./launch_MSA_test.py

(o,o) ---|Welcome to this program for Multiple Trace Alignment !!
< . /
...".-...
=====
MULTI ALIGN
- Number of sequences: 500
=====
Temps d'exécution: 11.21456795299946 secondes.
florent@florent-Lenovo-Legion-5-15ARH05: ~/Documents/N2/AlgoMMA/V3/ProjectTraceON
-main$
```

(b) Exemple de résultat en exécutant les deux fichiers Python avec un nombre de séquence de 50 de taille 500.

FIGURE 5

```
Quatre 1 1
Test_output.txt
~\Bibliothèque personnelle\ProjectTraceON\Outils\

Taille: 19
Nom Seq: TAILLE SEQUENCE
Trace_101 23 E1 E1 . . . . E4 . . . . E5 E7 . . . . S
Trace_111 39 E1 . . . . E4 . . . . E7 . . . . S
Trace_121 39 E1 . . . . E4 . . . . E7 . . . . S
Trace_131 25 E1 E2 E1 . . . . E4 . . . . E7 . . . . S
Trace_141 69 E1 E2 E1 E1 E1 . . . . E4 . . . . E7 . . . . S
Trace_151 63 E1 E1 E2 E2 E2 E4 . . . . E4 . . . . E7 . . . . S
Trace_161 42 E1 E2 . . . . E7 . . . . E2 . . . . E1 E4 . . . . S
Trace_171 55 E1 . . . . E1 E2 . . . . E4 . . . . E4 . . . . E4 . . . . S
Trace_181 61 E1 . . . . E1 E2 . . . . E4 . . . . E4 . . . . E7 . . . . S
Trace_191 53 E1 . . . . E1 E2 . . . . E4 . . . . E4 . . . . E7 . . . . S
Trace_141 77 E1 . . . . E1 E2 . . . . E4 . . . . E4 . . . . E7 . . . . S
Trace_151 10 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_161 39 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_171 74 E1 . . . . E1 E1 . . . . E2 . . . . E4 . . . . E5 . . . . E7
Trace_181 20 E1 . . . . E1 E1 . . . . E7 . . . . E7 . . . . S
Trace_191 77 E1 . . . . E1 E2 . . . . E4 . . . . E4 . . . . E4 . . . . E4
Trace_101 81 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_111 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_121 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_131 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_141 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_151 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_161 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_171 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_181 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Trace_191 40 E1 . . . . E1 E1 . . . . E4 . . . . E7 . . . . E7 . . . . S
Score: 608
```

FIGURE 6 – Exemple de sortie d'alignement multiple sur un jeu de données de 19 séquences (fichier "Test\_output.txt").