Multiple Sequence Alignment Project

Damien Garcia – Florian Echelard

January 2023

Contents

List of Figures

List of Tables

1 Compilation and execution

In order to facilitate usage of the program, a Makefile is provided in the projects archive allowing for easy compilation of the diverse programs:

- **`make all`** : Compiles the complete project.
- **`make clean`** : Removes all the compiled files and potential precompiled headers.
- **`make rebuild`** : Combination of **`make clean`** and **`make all`**
- **`make data_generation`** : Compiles the traces generation program.
- **`make alignment`** : Compiles the alignment program.
- **`make quality_analysis`** : Compiles the scoring program.

Six example are at disposal to test the program and check the execution process. The program runs syntaxic and semantic checks on the expression to avoid errors and unpredictable behavior during traces generation. Working examples and basic syntaxic/semantic error case can be executed with the commands:

- **`make test_simple`** : Working trace generation using simple generative regions.
- **`make test_complex`** : Working trace generation using complex generative regions.
- **`make test_semantic[1-4]`** : Execution returns syntaxic and semantic errors handled by the data_generation program[1].

Each section of the project can be executed independently with the same command line structure: **`./main [inputFile] [outputFile]`**. Input file is either the parameters file for the data_generation program, the traces to align for the alignment program, or the aligned traces for the quality_analysis program. Output file is the path where the results should be written.

A Python3 script is also at disposal for easy and efficient complete execution of the project. Use the command: **`python3 pipeline.py [pathToDatasetDirectory]`**. The pipeline will handle any necessary compilation, execute the program accordingly to previously explained process and create the results directories if necessary.

2 Traces generation

The first section of the project aims at generating sequences called traces which are composed of tops and events. Events can be anchors, meaning they are shared by all generated traces and only exist once for each trace, or simple events that depending on the generation parameters, either exist in a trace or not, and could in some cases have repetitions. For easier further analysis of the traces, tops are represented by a dot (“.”), events always start by a full case “E” for anchors and lower case “e” for other events.

In order to generate traces, the program relies on parameters files providing a RegEx-like[2] sequence, the number of traces to generate and the maximal size of the traces to generate.

[1] `make test_semantic4` is currently not working. In theory it should return an error, yet it generates traces.
[2] Regular Expression

Listing 1: Parameter file structure

```
# GENERATION PARAMETERS
expression=(2-3)E1<(4-8)e2X10|E2%25>E2(2-4)E3<(2-5)e1%40>
number_of_traces=20
maximum_length=100
```

The expression is separated in 3 sections types :

- Simple generative regions : Containing only tops, delimited by parenthesis.
- Complex generative regions : Containing tops and events, delimited by a less-than and greater-than signs.
- Anchors : Events outside of generative regions.

Table 1: Generative regions format

Expression	<	(	min	-	max	)	+	event	X	val	|	...	>
Required for simple generation		■	■	■	■	■							
Required for complex generation	■	■	■	■	■	■		■	■	■			■

Traces are generated and written in a single file, with on expression per line as seen below.

Listing 2: Traces generated from expression (2-3)E1<(4-8)e2X10|E2%25>E2(2-4)E3<(2-5)e1%40>

```
. . . . E1 . e2 . e2 E2 . . . . . E3 . . . . .
. . E1 . . e2 e2 . e2 . . E2 . . E3 e1 . e1 e1 e1
. . E1 . . . . . . . E2 . . . . . E3 . e1 e1
. . . . E1 e2 . e2 . . . . E2 . . E3 . . . . . .
. . . . E1 e2 . e2 . E2 . . . . . E3 . . .
. . . E1 . . . . . . . . . . . . E2 . . . . . E3 . .
. . . E1 . . . e2 . . e2 . . E2 . . . . E3 . . . . .
. . . . E1 . . e2 e2 . . . . . E2 . . . E3 . e1
. . E1 . e2 . . . e2 E2 . . E3 . . . e1
. . E1 . . . . . . . . . E2 . . E3 . e1 .
. . . . E1 e2 . . . . . . . e2 . . E2 . . E3 . . . . .
. . E1 . . . . . E2 . . . . . E3 . e1
. . . E1 . . . . . . . . . E2 . . . . . E3 . . .
. . E1 e2 e2 . . . . . . . E2 . . E3 . .
. . E1 . . . . . . . E2 . . . . E3 . e1 . e1 e1 e1
. . . . E1 . . . . . . . . E2 . . . . . E3 . e1 e1
. . . . E1 e2 . . . . . E2 . . . . . E3 . . .
. . . E1 . . . . . . . e2 E2 . . E3 . . . .
. . . E1 e2 . e2 e2 E2 . . . . . E3 . . .
. . . E1 . . e2 e2 e2 . . . E2 . . E3 . .
```

3 Multiple Sequence Alignment (MSA)

To perform the MSA, the program relies on two main methods namely (i) pairwise alignment algorithm and (ii) Unweighted Pair Group Method with Arithmetic mean (UPGMA) algorithm. The decision guiding the order in which traces will be aggregated by UPGMA algorithm is by measuring the dissimilarity between each pair of traces. This measure is inherently linked to the pairwise alignment process.

Step 1 At first, all the traces are stored into a data type that can contain one or more traces. This way, when a pair of traces will be aggregated, the algorithm won't have to rely on different methods and functions depending on the progress of the MSA. All functions and methods are programmed to process two single traces as well as batches of traces.

Step 2 Next, every pair is aligned which allows to get the dissimilarity score. The pair with the lowest dissimilarity score is then selected and modified considering the alignment results. At this point the traces probably include gaps represented by hyphens ("-"). The updated traces are then aggregated into one batch of traces.

Step 3 The dissimilarity matrix[3] is updated by removing the pairs and adding the new batch of traces. The dissimilarity score for the new batch is then calculated with every other traces and/or batches. This method allows the algorithm to avoid recalculating every dissimilarity score by keeping unmodified data and only calculating scores for new batches.

Steps 2 and **3** are repeated until all traces are aggregated into one batch.

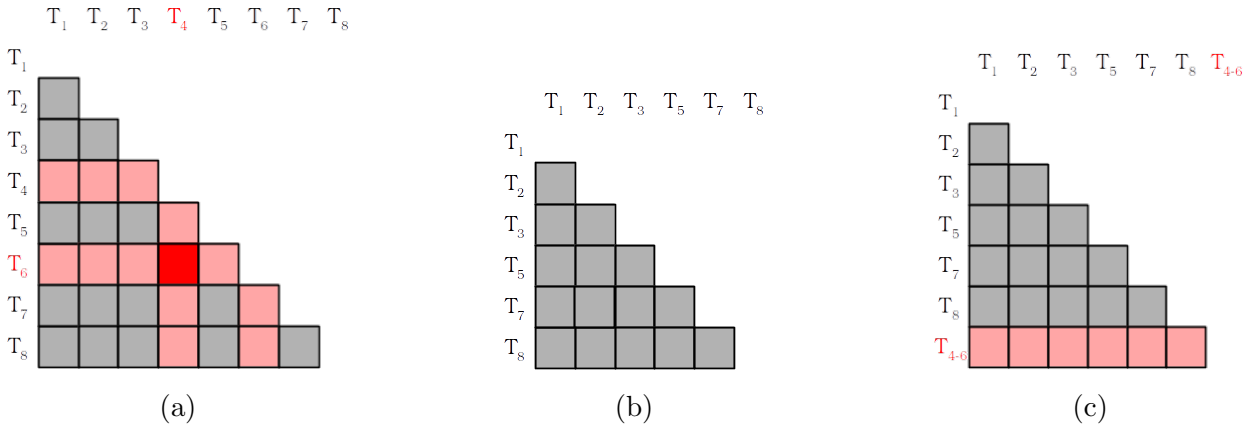


Figure 1: Dissimilarity matrix update
(a) Selecting minimum dissimilarity (red cell) and removing the values from the corresponding traces (pink cells). *(b)* Resulting matrix after step *a*. *(c)* Adding aggregated traces and performing dissimilarity calculations for the new cells.

At the end of program execution, the result is shown in a formatted console output to check the alignment (Sup.Fig. 2). The result is also returned to the output file at the path provided at program execution.

4 MSA quality assessment

- **score_e** : Counts pairwise events and/or tops matches in the MSA.
- **match_e** : Counts pairwise events matches in the MSA.
- **score_g** : Counts the gaps in the MSA.
- **proj_length** : Length of the traces on the MSA (with gaps).
- **general_score** : $\frac{score\_e \times 0.1 + score\_g \times 0.9}{nbPosition}$ with $nbPosition = length_{traces} \times number_{traces}$

[3]Represented by a lower triangular matrix for optimization purpose. $Dissim(T1, T2) = Dissim(T2, T1)$

In order to evaluate the quality of the Multiple Sequence Alignment, we created scoring functions and generated plots (Sup.Fig. 4) based on these functions. The first notable remark is that the program execution time seems to be proportional to the square of traces lengths (Sup.Fig. 4a). This result was expected, considering that the algorithm implemented is a variation of the pairwise alignment of sequences which has a complexity of $O(nm)$. n and m being of relative similar length, the complexity could be considered quadratic $O(n^2)$.

The quantity of top against top and event against event matches seems to be linearly proportional to the traces length (Sup.Fig. 4c). This result wasn't as much expected but hoped for. One interpretation is that the augmentation in length does not seem to impact the overall alignment by introducing gaps in the traces. Another way of interpreting this is that the cost of gap introduction in traces seems to be relatively well chosen, maintaining tops and events aligned respectively with other tops and events.

Concentrating only on event matches is not as straightforward (Sup.Fig. 4d). The limited dataset produced for analysis doesn't denote a clear statistical trend in the evolution of event matches. Checking the aligned traces one by one exposes another behavior of the algorithm: when the generated traces have significant length variations between them, the cost of introducing many gaps becomes higher than creating mismatches (Sup.Tab. 4). This will by definition, lower the score of the *match_e* function.

We also created a global score to assess the overall quality of MSA (Sup.Fig. 4b). Although variations are low in the datasets generated for quality assessment, it seems that the algorithm performs better with traces between 30 and 60 elements. Below 60 elements, whatever the inherent complexity of traces, the execution time is kept lower than 50 milliseconds. Above this threshold, execution time starts to rapidly increase going to ≈250 milliseconds for traces over 100 elements (Sup.Fig. 4a).

5 Prospects

Overall the project was build as presented in requirements specification, although slight modifications on scoring functions were made to better assess the quality of Multiple Sequence Alignment. These modifications allowed us to understand the strength and weaknesses of the algorithm and fit costs of mismatches or gap insertions.

As seen by the various results presented, the alignment is of comparatively good quality considering the questionable methods used in the algorithm. Multiple optimizations and usage of more precise but time consuming methods could have been implemented.

The main concern is that the values for indel[4] and substitutions are fixed *a priori* while these weights could be carefully chosen based on the inputs. For example, with batches of traces showing large difference between lengths, the number of gaps introduced will accordingly be higher than in an ideal case. Therefore, lowering the cost of gaps in comparison to the cost of mismatches on events should allow for wider gaps and align tops and events. Another way of solving this problem could be to lower the cost of gap introduction around other gaps, also resulting in wider gap segments and better alignment of tops and events.

The second concern is that this algorithm aims at aligning traces which have no actual real life meaning. The traces could be the reflection of time series, genes recognition in a sequencing process, or any other situation. Adding context around these traces would allow for revisions on the alignment process.

[4]Insertions and deletions

Supplementary material

Code accessible on GitHub repository: https://github.com/Damien-Garcia-Bioinformatics/aap_mma_project

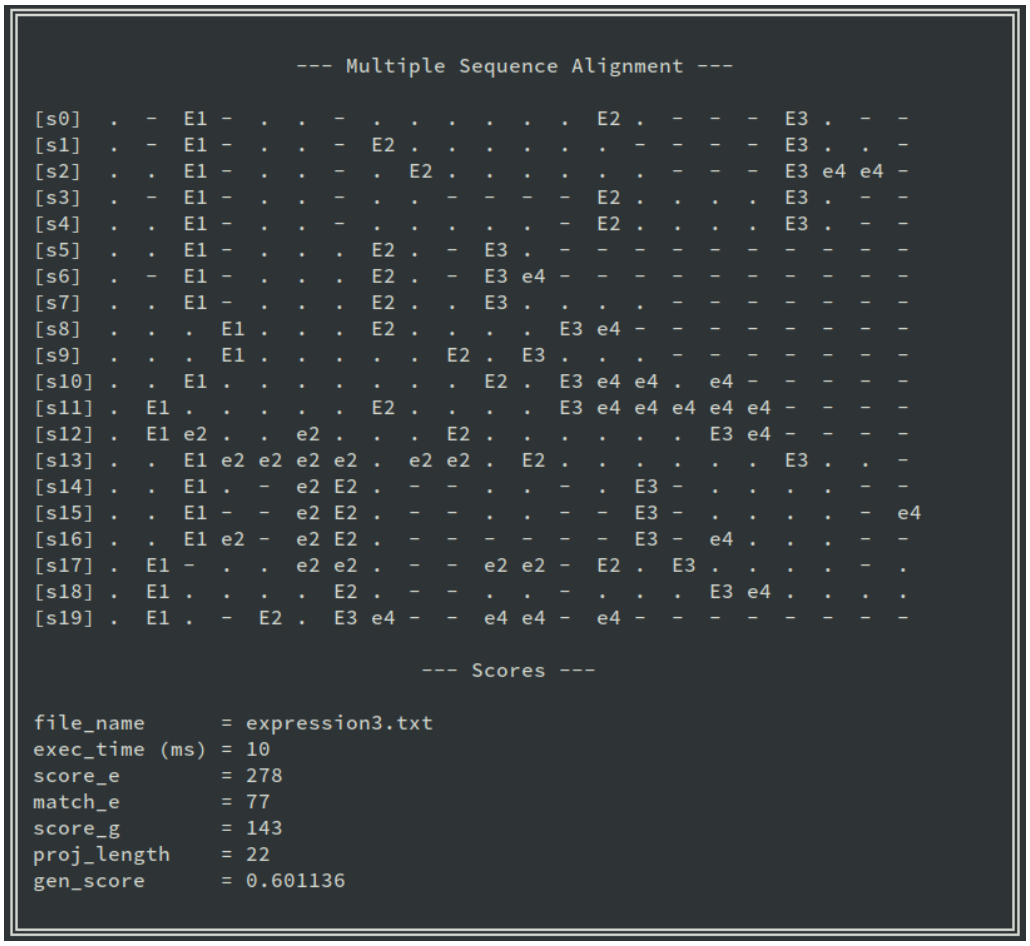


Figure 2: Terminal output from python pipeline

Figure 3: Datasets directory structure

Table 2: Dataset 1 scoring results: based on expressions containing simple generative regions.

file_name	**exec_time** (ms)	**score_e**	**match_e**	**score_g**	**proj_length**	**general_score**
expression1.txt	13	462	58	194	33	0.659394
expression2.txt	40	812	79	425	62	0.623629
expression3.txt	20	589	55	185	39	0.703333
expression4.txt	12	427	78	188	31	0.650161
expression5.txt	28	726	53	267	50	0.680100

Table 3: Dataset 2 scoring results: based on expressions containing complex generative regions.

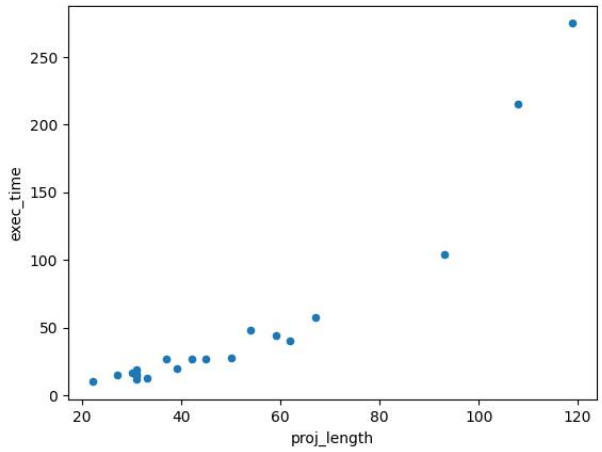
file_name	**exec_time** (ms)	**score_e**	**match_e**	**score_g**	**proj_length**	**general_score**
expression1.txt	19	405	103	202	31	0.620484
expression2.txt	15	399	109	130	27	0.689074
expression3.txt	10	278	77	143	22	0.601136
expression4.txt	27	580	138	154	37	0.726216
expression5.txt	16	451	80	164	31	0.681129

Table 4: Dataset 3 scoring results: based on expression with high length variability.

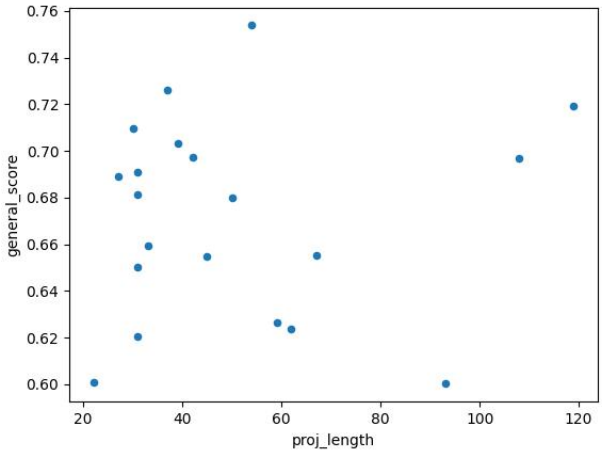
file_name	**exec_time** (ms)	**score_e**	**match_e**	**score_g**	**proj_length**	**general_score**
expression1.txt	104	1169	180	644	93	0.600269
expression2.txt	215	1615	401	513	108	0.696667
expression3.txt	275	1847	487	495	119	0.719244
expression4.txt	17	459	103	126	30	0.709500
expression5.txt	14	459	108	152	31	0.690806

Table 5: Dataset 4 scoring results: based on extremely variable expressions containing both simple and complex generative regions and multitude of events

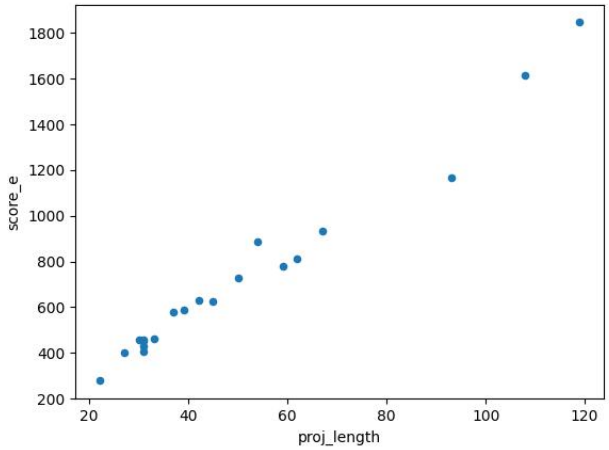
file_name	**exec_time** (ms)	**score_e**	**match_e**	**score_g**	**proj_length**	**general_score**
expression1.txt	58	933	176	386	67	0.655448
expression2.txt	44	779	186	383	59	0.626610
expression3.txt	27	628	152	206	42	0.697381
expression4.txt	27	626	142	258	45	0.654667
expression5.txt	48	885	157	176	54	0.753796



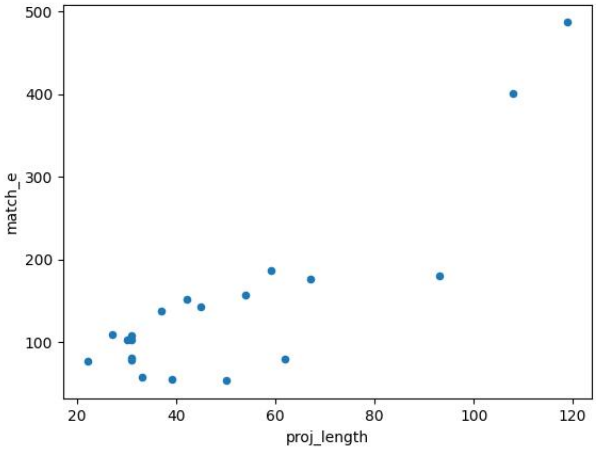
(a) Time of MSA algorithm execution



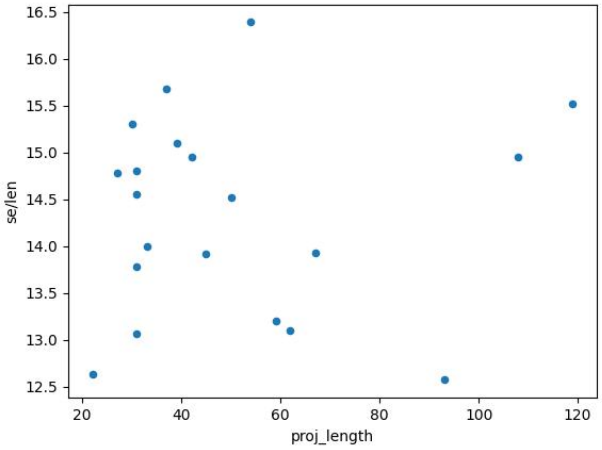
(b) Normalized quality of alignment



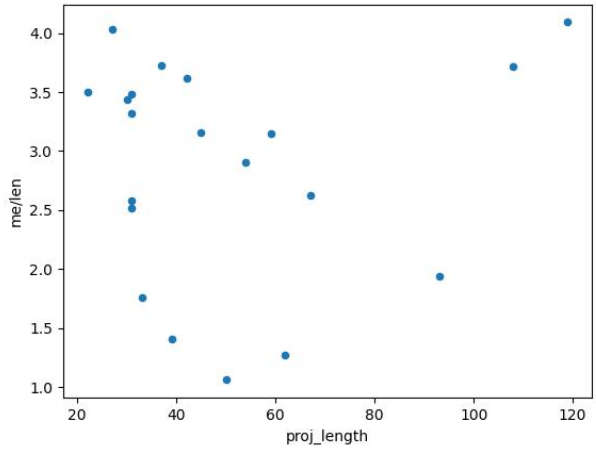
(c) Number of pairwise tops and/or events matches



(d) Number of pairwise events matches



(e) Number of pairwise tops and/or events matches normalized by traces length



(f) Number of pairwise events matches normalized by traces length

Figure 4: Scores of alignment quality assessment functions relative to traces lengths