Multiple Sequence Alignment Project

Damien Garcia – Florian Echelard

January 2023

Contents

1 Traces generation **2**
1.1 Objectives . . . 2
1.2 Compilation and execution . . . 2

2 Multiple Sequence Alignment (MSA) **3**
2.1 Compilation and execution . . . 3

3 MSA quality assessment **3**
3.1 Scoring functions . . . 3

List of Figures

List of Tables

1 Generative regions format . . . 2

1 Traces generation

1.1 Objectives

The first section of the project aims to generate 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[1] sequence, the number of traces to generate and the maximal size of the traces to generate.

```
1 # GENERATION PARAMETERS
2 expression=(2-3)E1(10-12)E9(1-4)E3
3 number_of_traces=20
4 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	×	×	×	×	×	×		×					×

1.2 Compilation and execution

Compile the program using Makefile command:

- **`make data_generation`**

The program runs syntaxic and semantic checks on the expression to avoid errors and unpredictable behavior during traces generation. Working examples can be executed with the commands:

- **`make test_simple`** – shows working trace generations with simple generative region.
- **`make test_complex`** – shows working trace generations with complex generative region.
- **`make test_semantic[1-4]`** – shows syntaxic/semantic errors[2].

[1] Regular Expression

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

2 Multiple Sequence Alignment (MSA)

To perform the MSA, the program relies on two main methods being (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 either 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, modified with considerations to the alignment results. At this point the traces might 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.

2.1 Compilation and execution

Compile the program using Makefile command:

- **`make alignment`**

3 MSA quality assessment

3.1 Scoring functions

[3]Represents a lower triangular matrix for optimization purpose. Dissim(T1, T2) ≡ Dissim(T2, T1)