Advanced Human Machine Interaction
Interaction Data Analysis
TDM02 Sequence Alignment

INSA Rouen Normandie and Université de Rouen Normandie

Purpose and organisation

This practical session studies local sequence alignment as a method for extracting similar fragments from discrete interaction traces. You will first formalise the problem, then implement Smith–Waterman and analyse how modelling choices affect the result.

You may use the programming language of your choice. Python starter files are provided.

Provided files

- `student_skeleton.py`: starting point for Smith–Waterman;
- `sequence_generator.py`: reproducible trace generator;
- `experiment_skeleton.py`: guided analysis of generated traces.

Learning objectives

- formalise a pattern extraction problem before selecting a method;
- implement and validate a local sequence alignment algorithm;
- explain the effects of match, substitution and gap scores;
- identify how the representation and the data generation process influence the patterns found.

Part 1 Problem formalisation

An interactive system records one event each time a user performs an action. A trace is therefore a finite sequence over an alphabet A. We want to find similar behavioural fragments in two traces, even if one fragment contains a substituted, missing or additional event.

Before considering an algorithm, specify the following elements.

1. Define the input traces and their representation.
2. Define the expected output. Include both the aligned fragments and their positions in the original traces.
3. Identify the problem class.
4. List the parameters that express what counts as a plausible behavioural variation.
5. Propose at least two ways to evaluate the result.

Part 2 Smith–Waterman

Use the following scoring model:

Operation	Score
Identical events	$+2$
Substitution	-1
Gap	-2

Let $X = (A, B, C)$ and $Y = (A, X, C)$. Fill the Smith–Waterman matrix. The first row and column contain zeros.

	–	A	X	C
–	0	0	0	0
A	0			
B	0			
C	0			

For each cell, use

$$H[i, j] = \max\{0,\ H[i-1, j-1] + \mathrm{sub}(x_i, y_j),\ H[i-1, j] + g,\ H[i, j-1] + g\}.$$

1. Detail the computation of $H[1, 1]$, $H[2, 2]$ and $H[3, 3]$.
2. Identify a maximal value and perform the traceback.
3. Give the aligned fragments, their score and their intervals in X and Y.
4. Explain why the recurrence includes zero.

Part 3 Implementation and tests

Implement the following function. You may adapt the types and return structure to your language.

```
def smith_waterman(sequence_a, sequence_b,
                   match=2, mismatch=-1, gap=-2):
    """Return the score matrix and one best local alignment."""
```

The result must contain:

- the complete score matrix;
- one maximal score;
- both aligned fragments, including gaps;
- start and end positions in each source trace. Use half-open intervals $[start, end)$.

Suggested implementation order:

1. initialise a $(|X| + 1) \times (|Y| + 1)$ zero matrix;
2. fill the matrix and remember the position of a maximal value;
3. trace back while the current value is positive;
4. reverse the two alignments produced during traceback;
5. return the result without printing it inside the function.

Validate your implementation with the following tests.

Sequence A	Sequence B	Expected best score
1 2 3 4	1 2 9 3 4	6
6 7 8 9	6 7 8 9	8
1 1	9 9	0

Also check that exchanging the two input traces preserves the maximal score.

Part 4 Influence of the scoring model

Use $X = (1, 2, 3, 4)$ and $Y = (1, 2, 9, 3, 4)$. Compare at least the following settings.

Match	Substitution	Gap	Best score	Interpretation
$+2$	-1	-1		
$+2$	-1	-2		
$+2$	-1	-4		
$+2$	-3	-2		

1. Which settings interpret event 9 as an insertion? Which settings favour a substitution or a shorter local fragment?
2. What assumptions about interaction traces does a strong gap penalty express?
3. Would you use the same scores for keyboard events, noisy gesture recognition and web page visits? Explain.
4. Propose a normalised score that makes alignments of different lengths easier to compare. State one limitation of your proposal.

Part 5 Generated traces and interpretation bias

The generator plants five motifs in synthetic interaction traces:

$$P_1 = (1, 2, 3), \quad P_2 = (3, 4), \quad P_3 = (6, 7, 8, 9), \quad P_4 = (7, 8, 7), \quad P_5 = (2, 4, 6).$$

Generate the datasets from the directory containing the supplied files:

```
python3 sequence_generator.py --output-dir data --seed 42
```

The generator creates four scenarios.

Scenario	Construction
`clean`	Exact motifs concatenated with no separator.
`separated`	Exact motifs separated by one to three unrelated events.
`corrupted`	Separated motifs with occasional substitutions, insertions and deletions.
`random_control`	Random traces with similar lengths and no planted motifs.

Each `.txt` file contains one trace per line. The corresponding `.jsonl` file also provides the planted motif identifiers and their intervals. Do not inspect the annotations before writing your hypotheses.

5.1 Hypotheses

Rank the four scenarios for each expected result and justify your answer:

1. highest local alignment scores;
2. easiest recovery of one complete planted motif;
3. largest number of alignments that cross motif boundaries;
4. highest risk of confusing chance similarities with planted patterns.

5.2 Experiment

Use the first 30 traces from each scenario and compute all unordered pairs. Complete the supplied experiment skeleton to report:

- the mean maximal score;
- the mean source span of the selected alignments;
- the proportion of alignments contained in occurrences of the same planted motif;
- the proportion of alignments that overlap several planted motifs.

Keep the scores from Parts 2–4. Inspect several alignments from each scenario before interpreting the aggregate values.

5.3 Interpretation

Answer the following questions.

1. Was the scenario with the highest scores also the easiest one for recovering individual motifs?
2. Why can concatenating perfect motifs create ambiguous or misleading alignments?
3. How can separator events make motif boundaries easier to recover even though they add noise?
4. Why is the random control necessary?
5. Which conclusions depend on the generator rather than on Smith–Waterman itself?
6. Propose one change to the generator and predict its effect before testing it.

Optional extension

Smith–Waterman explores the complete dynamic programming matrix. BLAST uses a seed-and-extend strategy to find promising local similarities faster in very large sequence databases. Explain which compromise BLAST makes between exhaustive optimality and computational cost. Would such a heuristic be useful for the interaction traces generated here?

Note

Thank you to ChatGPT for his help in the writing of this detailed practical session.