Bilkent University

Semester of Spring 2025

Department of Computer Engineering

CS481 Bioinformatics Algorithms

Homework Assignment #3: Multiple Sequence Alignment

via. pair-wise alignment and center-star alignment

Deniz Tuna Onguner

Bilkent ID: 22001788

Instructor: Can Alkan

18 April 2025, Friday

Discussion

Overview & Expectations

The aim of this assignment is to implement a multiple sequence alignment algorithm based on the center-star method with pairwise alignments. The algorithms are implemented in Python and the performance is evaluated in terms of execution time and memory usage. Memory usage is measured in terms of peak Python-level memory usage and resident set size (RSS) change after the call.

The test cases are generated randomly with a fixed sequence length of $L = 100$ and a fixed number of sequences $n = 32$, respectively.

The expected performance of the algorithm is as follows:

- The execution time is expected to be quadratic in the number of sequences n and quadratic in the sequence length L [1, 2].
- The memory usage is also expected to be quadratic in the sequence length L and linear in the number of sequences n [1, 2].

Execution Time Analysis

In Appendix, Figure 1 shows the execution time of the algorithm while increasing the number of sequences n with a fixed sequence length $L = 100$. Meanwhile Figure 2 shows the execution time while increasing the sequence length L with a fixed number of sequences $n = 32$.
In both cases, the blue curves track the dashed n^2 references almost perfectly. That is precisely what is expected, since center-star requires every pairwise alignment to score the central sequence in total of $\Theta(n(n\text{-}1)/2)$ alignments. With sequence length L fixed, each alignment costs $\Theta(L^2)$, giving total work of $\Theta(n^2L^2)$.

Memory Usage Analysis

In Appendix, Figure 5 shows the peak Python-level memory while increasing the number of sequences n with a fixed sequence length $L = 100$. Meanwhile Figure 6 shows the peak Python-level memory while increasing the sequence length L with a fixed number of sequences $n = 32$.
When the number of sequences n grows while the sequence length L is held constant, the centre-star implementation constructs one $(L + 1) \times (L + 1)$ dynamic-programming matrix for each of the n pairwise alignments. Although the matrices are released immediately after use, at peak the process still holds $\Theta(nL^2)$ cells in memory, so the resident-set size rises linearly with n.
Conversely, when n is fixed and L increases, each matrix expands quadratically in L. The log–log plot therefore shows an approximately unit-slope line in L^2-space: doubling L multiplies the memory footprint by four, yielding the same $\Theta(nL^2)$ bound.

Figures 3 and 4 (Appendix) report the change in RSS after each alignment call. Because RSS reflects pages reserved by the operating system rather than bytes actively used by the program, these plots are included only for completeness and are not interpreted in the discussion.

Conclusion

The plots confirm the textbook cost model, quadratic in the number of sequences and quadratic in sequence length in terms of execution time. The memory usage is also quadratic in sequence length and linear in the number of sequences, as expected.

Please note that I have utilized large language models (LLMs) to assist in writing this report. The LLMs were used to fix grammar and spelling errors, as well as to improve the overall readability of the text; but, I have reviewed and edited the content to ensure its accuracy and relevance. The final report reflects my understanding and interpretation of the assignment.

References

[1] Ilinkin, I., Ye, J., & Janardan, R. (2010). Multiple structure alignment and consensus identification for proteins. BMC bioinformatics, 11, 71. https://doi.org/10.1186/1471-2105-11-71

[2] Wang, L., & Jiang, T. (1994). On the complexity of multiple sequence alignment. Journal of computational biology : a journal of computational molecular cell biology, 1(4), 337–348. https://doi.org/10.1089/cmb.1994.1.337

Appendix

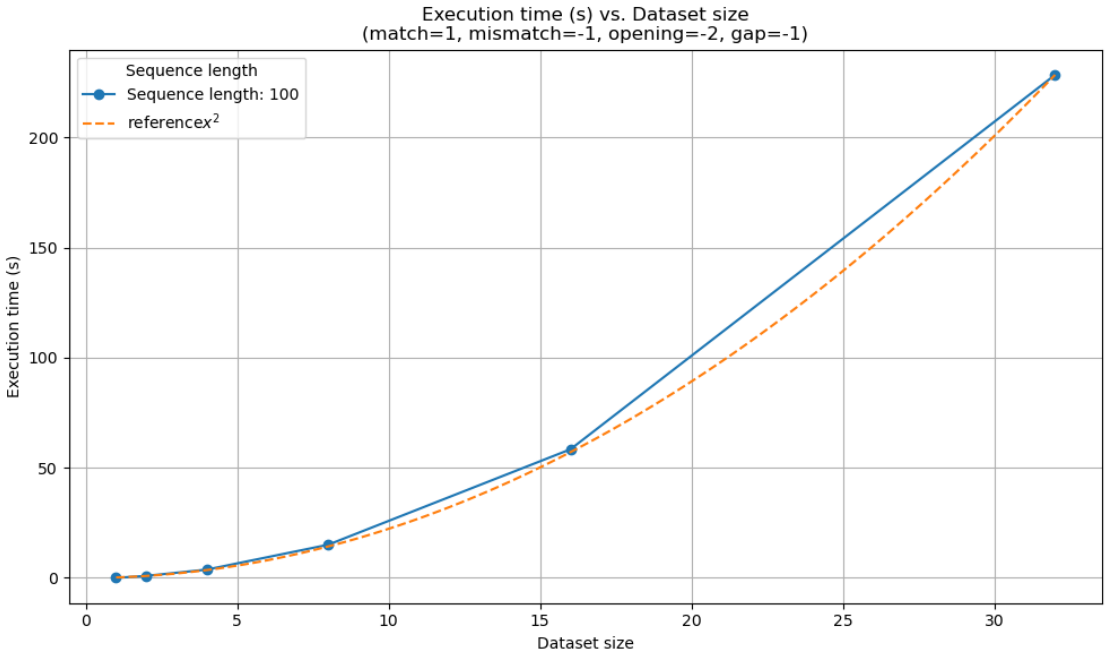

Figure 1: Execution time ($L = 100$) while increasing the number of sequences n.

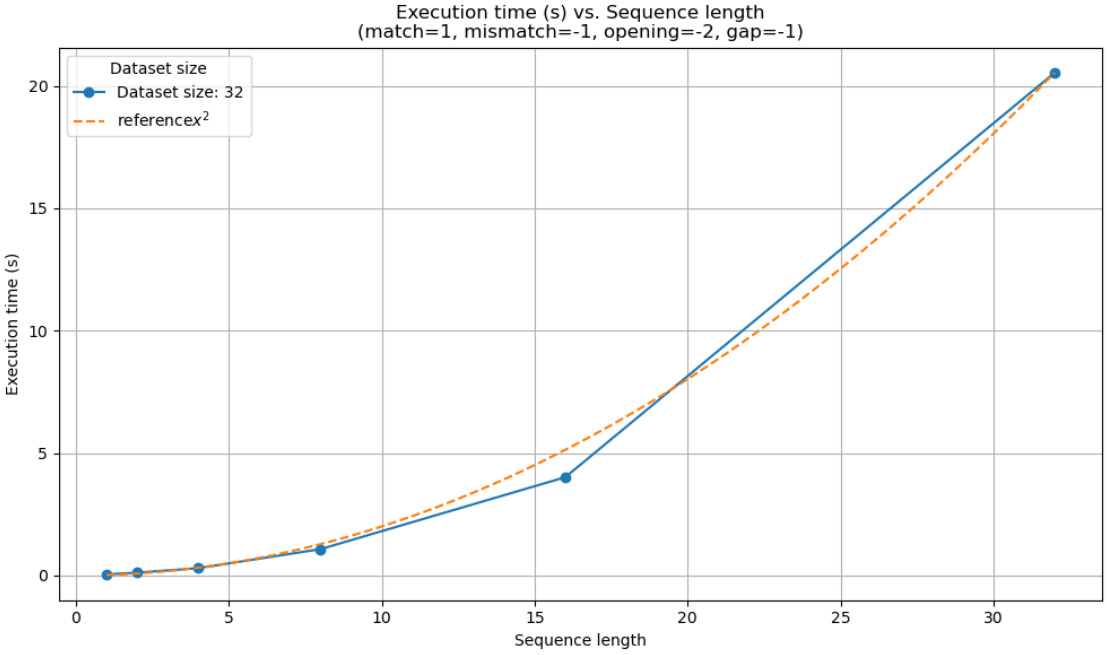

Figure 2: Execution time ($n = 32$) while increasing sequence length L.

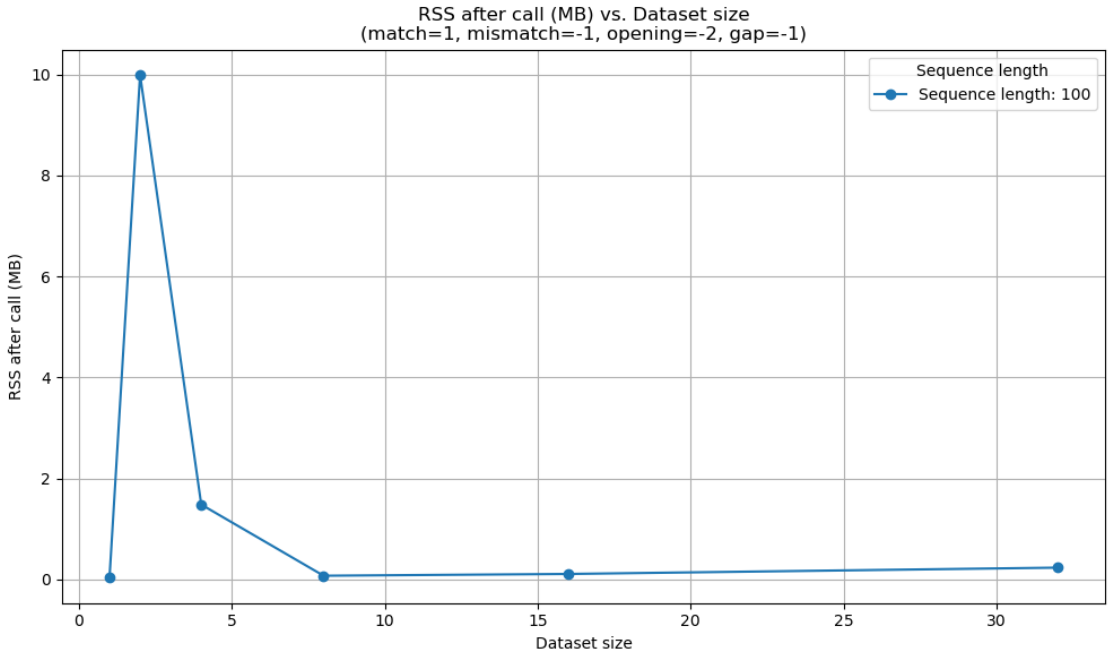

Figure 3: RSS change after call ($L = 100$) while increasing n.

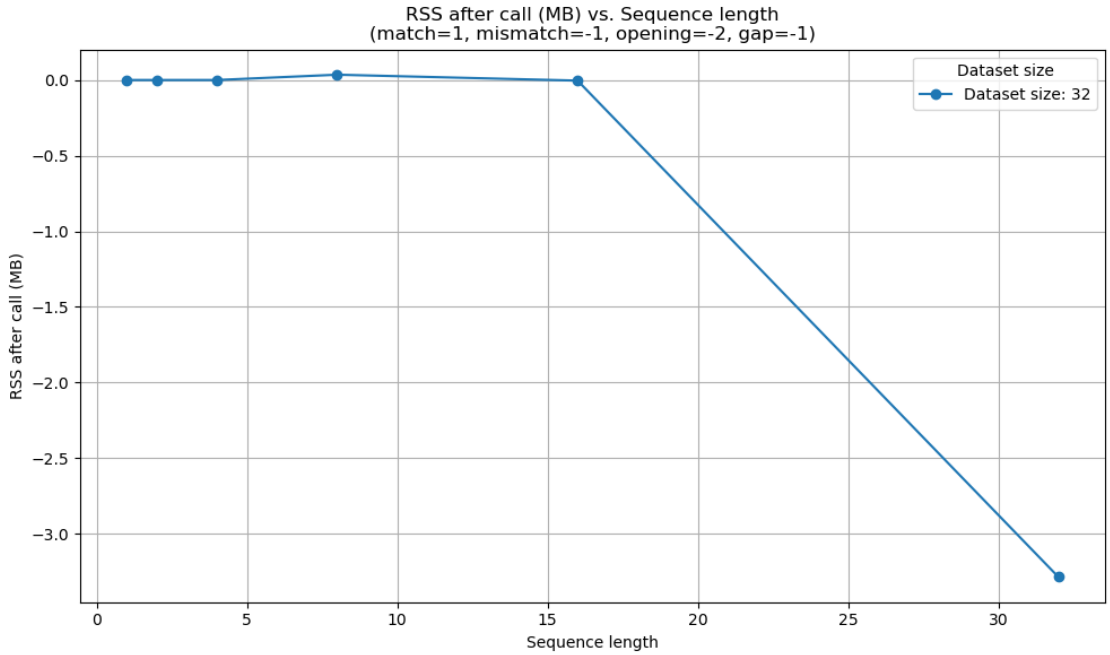

Figure 4: RSS change after call ($n = 32$) while increasing L.

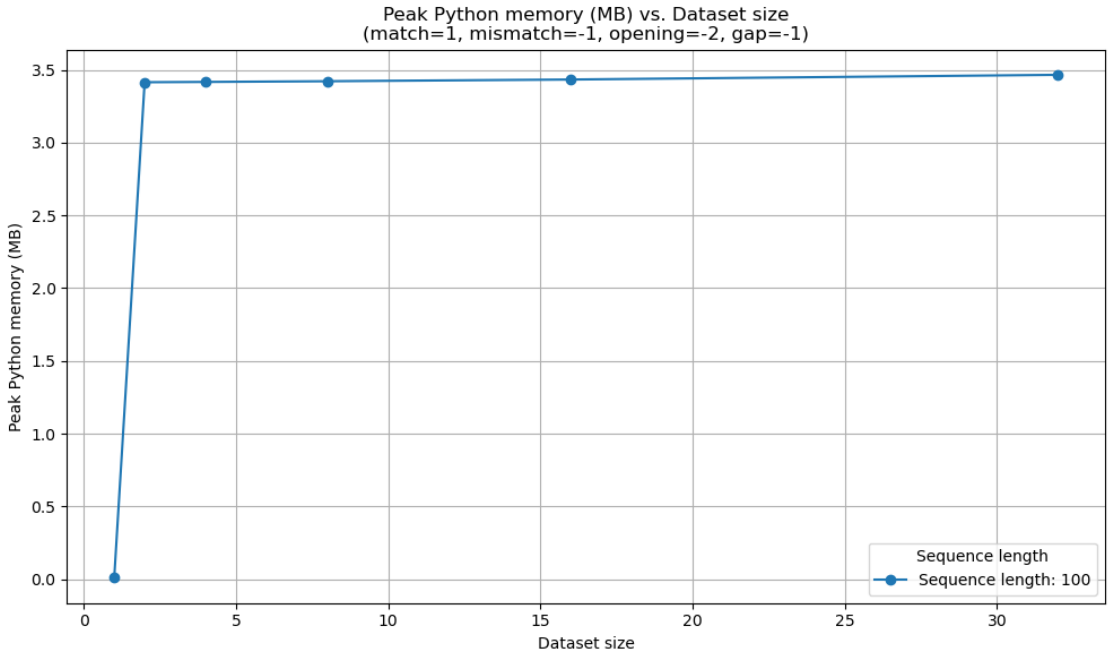

Figure 5: Peak Python-level memory ($L = 100$) while increasing n.

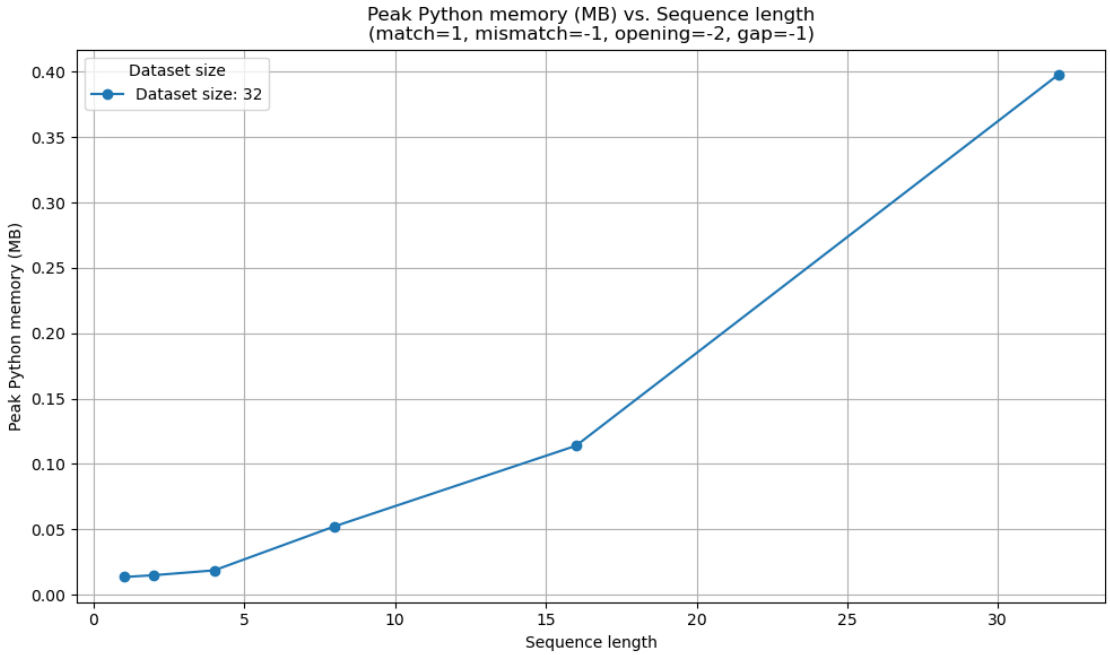

Figure 6: Peak Python-level memory ($n = 32$) while increasing L.