GapClean: Memory-Efficient Gap Removal for Large-Scale Multiple Sequence Alignments

Aarya Venkat[1]

***Abstract*— Multiple sequence alignments (MSAs) are fundamental to evolutionary analysis, yet large alignments often contain extensive gap-rich regions that obscure meaningful patterns and hinder visualization. We present GapClean, a memory-efficient tool implementing a novel 2D chunking algorithm for processing alignments exceeding available RAM. Using NumPy's vectorized operations, GapClean processes alignments in rectangular blocks, enabling gap removal on million-sequence datasets while maintaining $O(nm)$ time complexity with $O(rc)$ memory complexity, where r, c are user-defined chunk dimensions. The tool supports three filtering modes: threshold-based (percentage), reference-based (seed sequence), and entropy-based (Shannon entropy). Benchmarks demonstrate linear scaling across diverse alignment dimensions: the Pfam protein kinase family (PF00069, 1,051,876 sequences, 3,667 positions) processes in 35 seconds, while the GT2 glycosyltransferase family (PF00535, 157,052 sequences, 1,285 positions) completes in 11 seconds on standard hardware. GapClean is implemented in Python with minimal dependencies and installed via `pip install gapclean`.**

I. INTRODUCTION

Multiple sequence alignment (MSA) forms the foundation of comparative genomics, enabling evolutionary inference, functional annotation, and structural prediction[1, 2]. The Needleman-Wunsch algorithm[1] established dynamic programming as the standard approach for pairwise alignment, later extended to multiple sequences by tools like MUSCLE[2] and MAFFT[3]. However, modern MSAs frequently contain hundreds of thousands of sequences, creating visualization and analysis challenges.

Gap-rich columns in MSAs arise from insertions, deletions, or misalignments. While biologically meaningful in context, excessive gaps obscure conserved motifs and increase computational burden for downstream analyses. Existing tools either load entire alignments into memory (limiting scalability) or perform sequential processing (sacrificing speed).

We introduce GapClean, which addresses these limitations through a 2D chunking algorithm that decouples memory requirements from alignment size. By processing sequences and positions in rectangular blocks, GapClean enables gap filtering on arbitrarily large datasets with constant memory overhead.

II. ALGORITHM

A. Overview

GapClean processes MSAs in four stages: (1) FASTA normalization, (2) header-sequence separation, (3) 2D-chunked gap removal, and (4) output recombination. The core innovation lies in stage 3, detailed below.

[1]For Correspondence: Aarya Venkat (aaryakat@outlook.com)

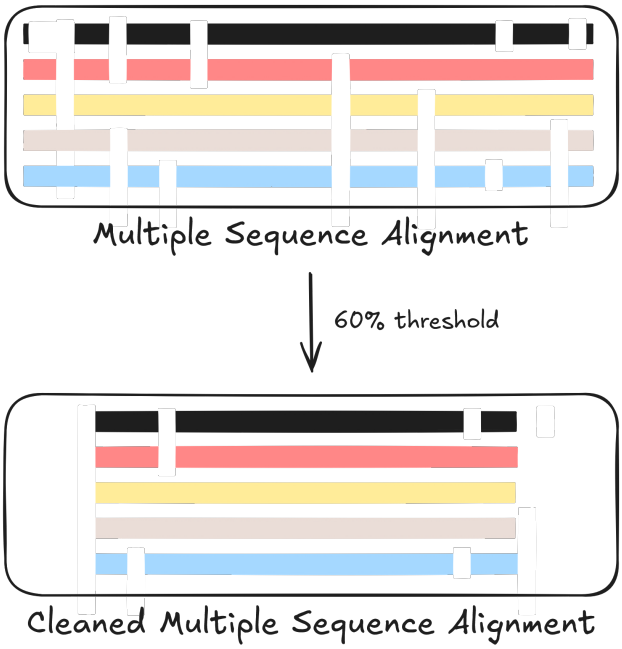

Fig. 1: GapClean removes gap-rich columns from multiple sequence alignments. Highly gapped regions often arise from lineage-specific insertions or alignment artifacts. Removing these columns focuses attention on conserved domains and motifs, improving alignment interpretability for downstream analyses.

B. 2D Chunking for Memory-Efficient Processing

Traditional approaches load the full $n \times m$ alignment matrix (where n is sequence count and m is alignment length) into memory, requiring $O(nm)$ space. For million-sequence alignments, this often exceeds available RAM. GapClean instead processes the alignment in rectangular blocks of size $r \times c$ (row-chunk $\times$ column-chunk), reducing peak memory to $O(rc)$ while maintaining $O(nm)$ time complexity.

Key advantages:

- **Memory independence:** Peak memory usage is $O(rc)$, independent of total alignment size
- **Cache efficiency:** Rectangular blocks improve CPU cache utilization
- **Vectorization:** NumPy operations[4] on byte arrays accelerate gap detection
- **Scalability:** Chunk sizes can be tuned for available memory

Algorithm 1 2D-Chunked Gap Removal

Input: n sequences of length m, chunk sizes (r, c), removal mode
Output: Filtered alignment
Initialize boolean mask $M \leftarrow [1]^m$
Initialize gap counter $G \leftarrow [0]^m$
for column-chunk $[c_s, c_e)$ in steps of c **do**
  for row-chunk $[r_s, r_e)$ in steps of r **do**
    for row $i \in [r_s, r_e)$ **do**
      $S \leftarrow \texttt{sequences}[i][c_s : c_e]$
      $A \leftarrow \texttt{np.frombuffer}(S, \text{dtype} = \text{uint8})$
      gaps $\leftarrow (A = 45) \vee (A = 46)$ {ASCII: '-' or '.'}
      $G[c_s : c_e] \leftarrow G[c_s : c_e] + \text{gaps}$
    end for
  end for
end for
Compute filtering criterion and update M
Apply M to filter sequences using second 2D pass
Return: Filtered sequences

C. Filtering Modes

Threshold mode: Remove columns where gap fraction exceeds threshold t:

$$M_j = \begin{cases} 1 & \text{if } G_j/n \leq t \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Seed mode: Remove columns containing gaps in reference sequence s:

$$M_j = \begin{cases} 1 & \text{if sequences}[s][j] \neq \text{'-'} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Entropy mode: Remove columns with Shannon entropy outside user-defined bounds. For column j with character distribution p_k:

$$H_j = -\sum_k p_k \log_2(p_k) \tag{3}$$

Users specify `--entropy-min` (retain variable regions) or `--entropy-max` (retain conserved regions).

III. IMPLEMENTATION

GapClean is implemented in Python 3.8+ with minimal dependencies (NumPy, tqdm). The pure-Python FASTA parser eliminates external tool dependencies, ensuring cross-platform compatibility. Default chunk sizes (5,000 × 5,000) balance memory efficiency with computational overhead, yielding ∼50 MB peak memory usage.

IV. BENCHMARKS

Experiments were conducted on an Apple M1 system using protein alignments from Pfam[5]. All benchmarks used default chunk sizes of 5,000 × 5,000 (rows × columns) and a 70% gap threshold for removal.

TABLE I: GapClean Performance Benchmarks (70% gap threshold)

Scale	Pfam	Seqs	Length	Time	Size
Tiny	PF15608	931	148	<0.1 s	215 KB
Small	PF00637	38,583	648	2 s	27 MB
Medium	PF00535	157,052	1,285	11 s	206 MB
Large	PF00069	1,051,876	3,667	35 s	7 GB

The protein kinase domain family (PF00069) with 1,051,876 sequences and 3,667 positions was reduced to 271 positions in 35 seconds, processing over 3.8 billion characters (7 GB raw alignment). The GT2 glycosyltransferase family (PF00535) shows efficient handling of broader alignments: 157,052 sequences with 1,285 positions reduced to 199 positions in 11 seconds (206 MB raw). The clathrin heavy-chain family (PF00637) with 38,583 sequences processes in 2 seconds (27 MB), while the RNA-binding domain family (PF15608, 931 sequences, 215 KB) completes in under 0.1 seconds, demonstrating negligible overhead for small datasets.

V. CONCLUSIONS

GapClean addresses the scalability challenge of MSA gap removal through memory-efficient 2D chunking. The tool enables analysis of datasets exceeding one million sequences on standard hardware. The implementation is lightweight, cross-platform, and integrates seamlessly into bioinformatics workflows.

AVAILABILITY

GapClean is released under the MIT License. Source code, documentation, and installation instructions are available at https://github.com/arikat/GapClean. The tool is installable via PyPI (`pip install gapclean`).

REFERENCES

[1] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[2] R. C. Edgar, "Muscle: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Research*, vol. 32, no. 5, pp. 1792–1797, 2004.

[3] K. Katoh, K. Misawa, K.-i. Kuma, and T. Miyata, "Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic Acids Research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[4] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[5] J. Mistry, S. Chuguransky, L. Williams, M. Qureshi, G. A. Salazar, E. L. Sonnhammer, S. C. Tosatto, L. Paladin, S. Raj, L. J. Richardson, *et al.*, "Pfam: The protein families database in 2021," *Nucleic Acids Research*, vol. 49, no. D1, pp. D412–D419, 2021.