






ReAlign-Star: An Optimized Realignment Method for Multiple Sequence Alignment, Targeting Star Algorithm Tools

Yixiao Zhai^{1,2}, Pinglu Zhang^{1,2}, Yi Liu^{1,2} , and Quan Zou^{1,2}  

¹ Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chengdu 611731, China
zouquan@nclab.net

² Yangtze Delta Region Institute (Quzhou), University of Electronic Science and Technology of China, Quzhou 324003, China

Abstract. In the star alignment algorithm for multiple sequence alignment, all sequences are aligned directly to the central ‘star’ sequence without using a guide tree. This method greatly reduces computation time, making star alignment-based tools effective for aligning homologous sequences with high similarity. However, as sequence similarity decreases or the number of sequences increases, the algorithm’s accuracy drops significantly. In particular, “junk sequences” with very low similarity to the central star sequence tend to result in poor alignments, which can degrade the overall alignment quality. While realignment methods can greatly enhance the accuracy of alignments, there is currently a lack of approaches specifically tailored for star alignment tools. This study presents ReAlign-Star, a realignment method specifically designed for star alignment-based tools. The core of ReAlign-Star employs two key strategies—filtering out “junk sequences” and applying local vertical partitioning for realignment—to efficiently improve the quality of star alignments. Experiments on both simulated and real datasets demonstrate that ReAlign-Star significantly improves alignment accuracy in most cases, outperforming the initial alignments and extending the applicability of star alignment tools. The source code and test data for ReAlign-Star are available on GitHub (<https://github.com/malabz/ReAlign-Star>).

Keywords: Multiple sequence alignment · Star-alignment · Post-processing · Realignment

1 Introduction

The star alignment algorithm, introduced by Stephen F. Altschul [1], is a fast MSA method that begins by selecting a central sequence and aligning other sequences to it. The pairwise alignments are then merged to form the final alignment. While it offers speed due to its direct alignment approach without a guide tree, it has key limitations: sequences that differ greatly from the central one can reduce accuracy, and alignment accuracy decreases as the number of sequences increases [2]. These issues stem from gap insertions during the merging stage, which degrade overall alignment quality, limiting the algorithm’s applicability despite its speed.

Many researchers have developed realignment methods to improve alignment quality based on initial alignments. These methods can be divided into three categories: horizontal partitioning, vertical partitioning, and hybrid approaches. Horizontal partitioning methods, such as ReAligner [3], Remove First (RF) [4], REFINER [5], and ReformAlign [6], use an iterative optimization strategy to gradually enhance alignment quality. In each iteration, the initial alignment is divided into two subsets, which are then realigned to generate a new alignment. If the new alignment shows improvement, it becomes the input for subsequent iterations until the objective function converges or the iteration limit is reached. TreeRefiner [7], on the other hand, improves alignment accuracy through three-dimensional alignment without iterative steps. Vertical partitioning methods, like Refin-Align [8], SpliVert [9], and RPFam [10], divide the initial alignment into multiple blocks, remove gaps within those blocks, and then re-align them. Blocks with higher scores replace the original ones, improving overall alignment quality. Hybrid methods, such as RASCAL [11], use tools like Secator to divide sequences into sub-families, identify global and local core blocks using the NorMD objective function, and realign poorly aligned regions. In star alignment, where all sequences are aligned to a single central sequence, the presence of low-similarity sequences or large numbers of sequences can greatly reduce alignment accuracy. Thus, after horizontal partitioning, gap errors may persist in the profiles, limiting the optimization potential of the initial star alignment.

Building on the ideas of vertical partitioning and hybrid methods, this study presents ReAlign-Star, a realignment method specifically designed for star alignment tools. We define “junk sequences” as those that consist primarily of bases, rather than gaps, in certain continuous regions compared to other sequences, as they poorly align with the rest. ReAlign-Star first analyzes the initial alignment to identify and remove these junk sequences. Then, the profile composed of the remaining sequences is locally realigned. Finally, ReAlign-Star aligns each “junk sequence” to this realigned profile using sequence-to-profile alignment, completing the realignment process. We rigorously evaluated ReAlign-Star on four simulated and three real nucleotide datasets. The results demonstrate that ReAlign-Star significantly improves most datasets’ average sum-of-pairs (aSP), Q, and TC scores, outperforming the initial alignments. This method not only improves the accuracy of star alignments but also broadens their applicability.

2 Materials and Methods

2.1 Overview of ReAlign-Star

Once the initial alignment from the star alignment tool is input into ReAlign-Star, the system first checks for the presence of “junk sequences.” Junk sequences are defined as those whose alignment with the central star sequence is of poor quality, characterized by a continuous string of bases in a region, while the other sequences consist entirely of gaps. The remaining sequences form a profile after identifying and removing junk sequences of the initial alignment. Then, the profile is locally realigned using a vertical partitioning method specifically designed for star alignment. Once the profile realignment is complete, ReAlign-Star aligns each junk sequence to the realigned profile, one

by one, using sequence-to-profile alignment. The process is complete when all junk sequences have been successfully aligned (Fig. 1A).

Filter Junk Sequences. In star alignment, each sequence is aligned pairwise with the

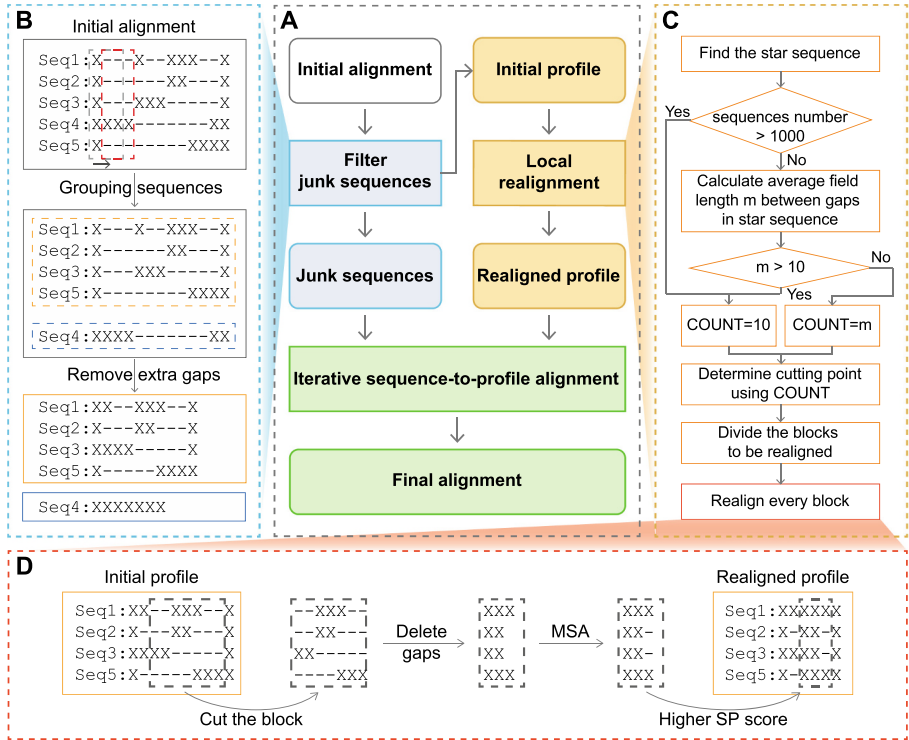


Fig. 1. The framework of ReAlign-Star. **A.** The details of the ReAlign-Star workflow. **B.** Schematic diagram of filtering junk sequences: A fixed-length sliding window (gray dashed box, starting position) moves from left to right across the initial alignment. In the red dashed box, Seq4 is detected and marked as a junk sequence. After filtering, the marked junk sequence (blue dashed box) is separated from the remaining sequences (yellow dashed box). Extra gaps are then removed, resulting in the final junk sequence (blue box) and the profile for realignment (yellow box). **C.** Flowchart of the local realignment process. **D.** An example illustrating the block realignment process.

central sequence, and these pairwise alignments are then combined to produce the final alignment. However, when some sequences have low similarity to the central sequence, their poor alignments can significantly reduce overall accuracy. To address this issue, ReAlign-Star introduces a sliding window mechanism to filter out “junk sequences.” The window, with a length of w (defaulting to 10 if not provided), spans w consecutive columns of the initial alignment. If a window contains only one sequence with all bases while the others consist entirely of gaps, that sequence is marked as a junk sequence. The reason for identifying junk sequences this way is that junk sequences are generated when

a sequence with low similarity to the central star sequence results in long insertions during alignment. If none of the other sequences have insertions in that region, the final merged alignment will show that only the junk sequence has bases aligned in that region, while the other sequences align with gaps. The window slides from left to right, continuing the screening process until it reaches the last column of the alignment. Marked sequences are classified as junk sequences, and all gaps in these sequences are removed. For the remaining unmarked sequences, columns made up entirely of gaps are deleted, resulting in a profile for local realignment (Fig. 1B).

Local Realignment of Star Alignment Based on Vertical Partitioning. ReAlign-Star employs vertical partitioning to analyze the distribution of gaps in the central (star) sequence, dividing the profile vertically and focusing on realigning gap-dense regions. The central sequence is user-specified, but by default, the sequence with the highest number of bases in the initial alignment is selected as the central sequence. The distance (in bases) between gap regions in the central sequence is measured, and the average, m , is calculated. *COUNT* is defined as the threshold for the maximum distance allowed between two adjacent gap regions. If the profile contains more than 1,000 sequences or if $m > 10$, *COUNT* is set to 10; otherwise, it is set to m . ReAlign-Star scans the central sequence, marking the start of a block whenever a gap ('-') is encountered and counting the non-gap bases. If the non-gap base count exceeds *COUNT*, the gap region is considered closed, and any block longer than 5 positions is recorded as valid for realignment. Once the central sequence has been scanned, the system cuts the profile into blocks based on the identified start and end positions (Fig. 1C).

After partitioning the blocks, ReAlign-Star realigns each block independently. This involves first removing all gaps within the block, then realigning the remaining sequences using an MSA tool. Once realignment is completed, the system compares the sum-of-pair (SP) scores before and after realignment (Eq. 1), selecting the result with the higher score as the final output and replacing the original block (Fig. 1D). This local realignment approach significantly improves the alignment quality in gap-rich regions while preserving the original alignment in uncut areas, avoiding unnecessary reprocessing of regions without gaps.

$$SP = \sum_{m=0}^L f(m) \quad (1)$$

The SP score is computed by summing the scores of L columns, where L represents the total number of columns in the block, and $f(m)$ denotes the score of the m -th column (Eq. 2).

$$f(m) = \sum_{i=0}^N \sum_{j \neq i}^N \text{score}(i, j) \quad (2)$$

Here, N is the total number of unaligned sequences, and $\text{score}(i, j)$ represents the score of the aligned pair in row i and row j . .. Matched pairs (identical letters) receive a score of 1, mismatches (different letters) get -1, letter-gap pairs are assigned -2, and all other cases score 0.

Implementation and Experimental Environment. ReAlign-Star is implemented in C++ 17 and is compatible with all UNIX-based platforms. It is freely available as open-source software under the MIT license and can be accessed at <https://github.com/malabz/ReAlign-Star>. All experiments were conducted on a server running Ubuntu Linux, equipped with a 2.7 GHz Intel(R) Xeon(R) Platinum 8168 processor and 1 TB of memory.

2.2 Datasets and Measurement

Seven nucleotide datasets were used in the experiment, comprising three real datasets and four simulated ones. The three real datasets include (1) a 23S rRNA dataset [12] with sequence lengths ranging from 1,909 to 3,485 bp, containing 641 mycobacterial 23S rRNA sequences from the SILVA rRNA database (<http://www.arb-silva.de/>); (2) a small dataset of 156 SARS-CoV-2 genomes with an average sequence length of 29,000; and (3) a large dataset of 24,310 SARS-CoV-2 genomes [13]. Additionally, one simulated dataset was generated through hierarchical tree simulations: (1) a human mitochondrial-like simulated dataset [14]; and (2) The CIPRES-simulated rRNA dataset covers evolutionary trees ranging from 128 to 4,096 taxa, all derived from a common ancestral rRNA sequence. Table 1 gives additional specific details.

To assess the alignment quality of the three real datasets, we employed the average sum of pairwise (aSP) score. This metric is calculated by dividing the SP score by the number of sequence pairs, using the same penalty parameters as in the local realignment section. In addition to the aSP score, we evaluated the alignment quality of the two simulated datasets using the Q score and TC score. Both scores were derived using the qscore program (<http://www.drive5.com/bench/>) to assess discrepancies between the alignment results and the reference alignment.

3 Results

3.1 Adaptive Block Division and Local Realignment Based on the Gap Distribution of the Central Sequence.

The central sequence's gap distribution typically reflects the overall gap pattern in the initial alignment, a feature crucial for local realignment. We used the default parameters of the widely adopted star alignment tool HAlign3 [15] to generate the initial alignments for all datasets and perform local realignment. To identify the blocks requiring local realignment, we calculated the distance D between adjacent gap regions in the central sequence of the initial alignment (i.e., the number of bases between two consecutive gap regions). Next, all gaps within the identified blocks were removed, and the blocks were locally realigned using HAlign3, MAFFT [16], and MUSCLE3 [17], respectively. The blocks with the higher alignment quality were retained as the final local realignment

results. To find the optimal threshold of D , we tested several values: 0, 2, 4, 5, 8, 10, and “Auto,” where “Auto” dynamically calculates the threshold based on the average distance between gaps in the central sequence (if the average exceeds 10, the Auto value is capped at 10). Additionally, to confirm the general applicability of the D threshold, we analyzed the average performance of the three MSA tools, demonstrating that the chosen threshold is optimal.

In the mt-like genome dataset, when D is set to 0, few or no blocks are identified for realignment, leading to the lowest aSP, Q, and TC scores. As D increases, both the number of identified blocks and valid blocks rise, steadily improving three evaluation metrics (Fig. 2D). For sequences with over 99% similarity, D change has minimal effect on block partitioning, and the impact on aSP and TC scores is negligible (Fig. 2A, C). Similarly, for sequences with over 96% similarity, D changes have little effect on Q scores (Fig. 2B). As sequence similarity decreases, the number of cut blocks increases significantly, leading to more valid blocks and, consequently, higher aSP, Q, and TC scores. Notably, when D is set to “Auto,” it results in the highest number of cut and valid blocks, leading to the best aSP, Q, and TC scores. In the CIPRES simulated RNA dataset, an increase in the number of sequences allows for a higher D , which enhances aSP and TC scores. Although the change in the Q score is relatively modest, it still shows significant improvement at larger D , especially in the dataset with 8,191 sequences (Fig. 3A-D).

In the 23S rRNA dataset, all values of D effectively identify blocks for realignment, with D set to “Auto” yielding the highest number of blocks and valid blocks, as well as the best aSP score after realignment (Fig. 4B). Interestingly, in the smaller SARS-CoV-2 genome dataset, all values of D result in the same number of cut blocks, but no valid blocks after realignment, indicating that local realignment does not enhance performance in high-similarity datasets with fewer sequences (Fig. 4E). Conversely, in the larger SARS-CoV-2 genome dataset, despite high sequence similarity, D set to “Auto” identifies the most blocks and improves SP, Q, and TC scores (Fig. 4H). These results indicate that when D is set to “Auto,” it can flexibly adapt to the gap distribution of different datasets.

3.2 Optimal MSA Tool Selection for Realignment Based on Dataset Size and Sequence Similarity.

Based on the above experiments, we calculated the average performance under all D thresholds to evaluate the applicability and performance differences of the three MSA tools under different threshold conditions.

In the hierarchical tree simulation dataset—mt-like genome (Fig. 2E-G)—each replicate containing 100 sequences, when sequence similarity was above 90%, the three MSA tools yielded comparable realignment results. However, when similarity dropped below 90%, and sequence length increased (the average lengths being around 16,000 bp for mt-like genome), MUSCLE3 consistently outperformed the others in terms of aSP, Q, and TC scores. MAFFT followed in performance, while HAlign3 lagged. In the CIPRES simulated RNA dataset, where sequence similarity was around 80%, and the sequence lengths were roughly the same, MUSCLE3 also led in aSP, Q, and TC scores, with MAFFT again outperforming HAlign3 (Fig. 3E-G).

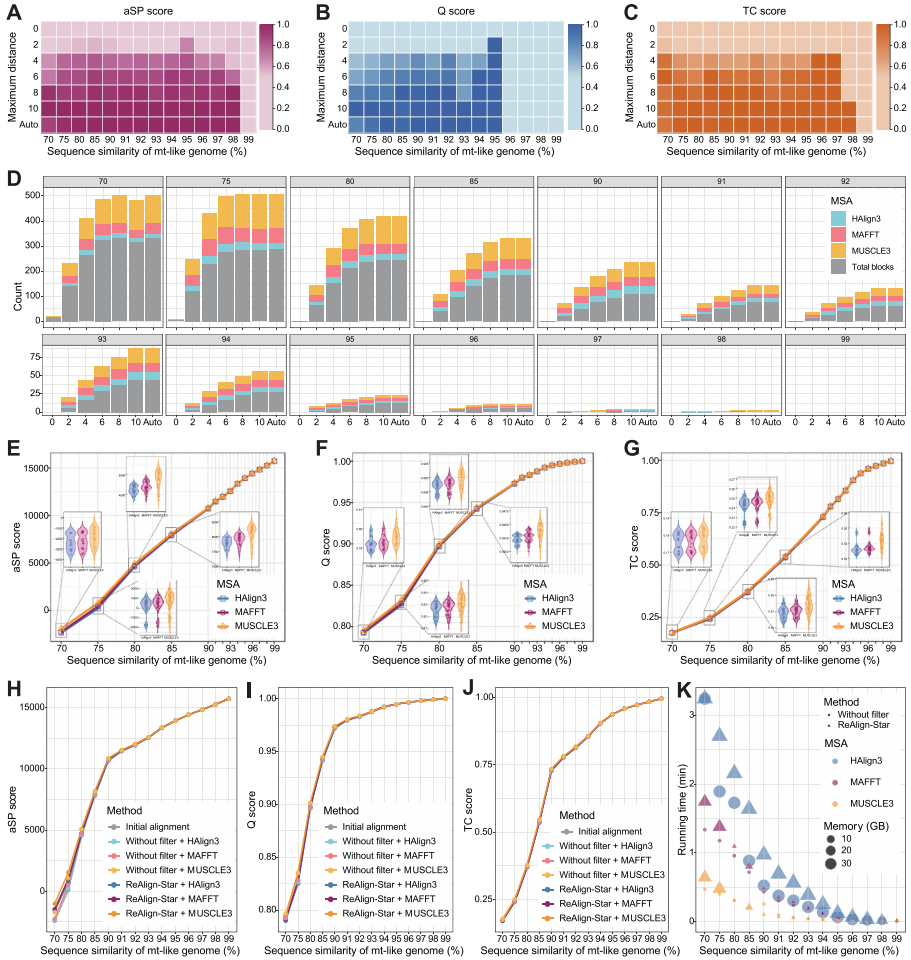


Fig. 2. Performance comparison based on the mt-like genome datasets. aSP (A), Q (B), and TC (C) scores for seven different maximum distances in the mt-like genome dataset. For each cell, the average score across 27 values (9 replicates \times 3 MSA tools) in a sub-dataset was calculated. Each column is then normalized using min-max normalization. D shows the average number of blocks identified across seven maximum distances under 14 different similarity conditions, along with the average number of blocks with improved SP scores after realignment using three MSA tools. Comparison of aSP (E), Q (F), and TC (G) scores for results from local realignment using HAlign3, MAFFT, and MUSCLE3 on the mt-like genome dataset. Each similarity condition includes nine replicates, with each data point in the line graph representing the average score of these replicates across seven different maximum distances. The box plots are based on a total of 63 values (9 replicates \times 7 thresholds). Comparison of aSP (H), Q (I), and TC (J) scores for ReAlign-Star with and without junk sequence filtering. K shows the running time and memory usage between two methods with three MSA tools. In each of the three line charts with dots, each point represents the average score of nine replicates. The bubble chart shows the average running time and memory consumption of nine replicates in each subset. Here, bubble height represents running time, and bubble size represents memory consumption.

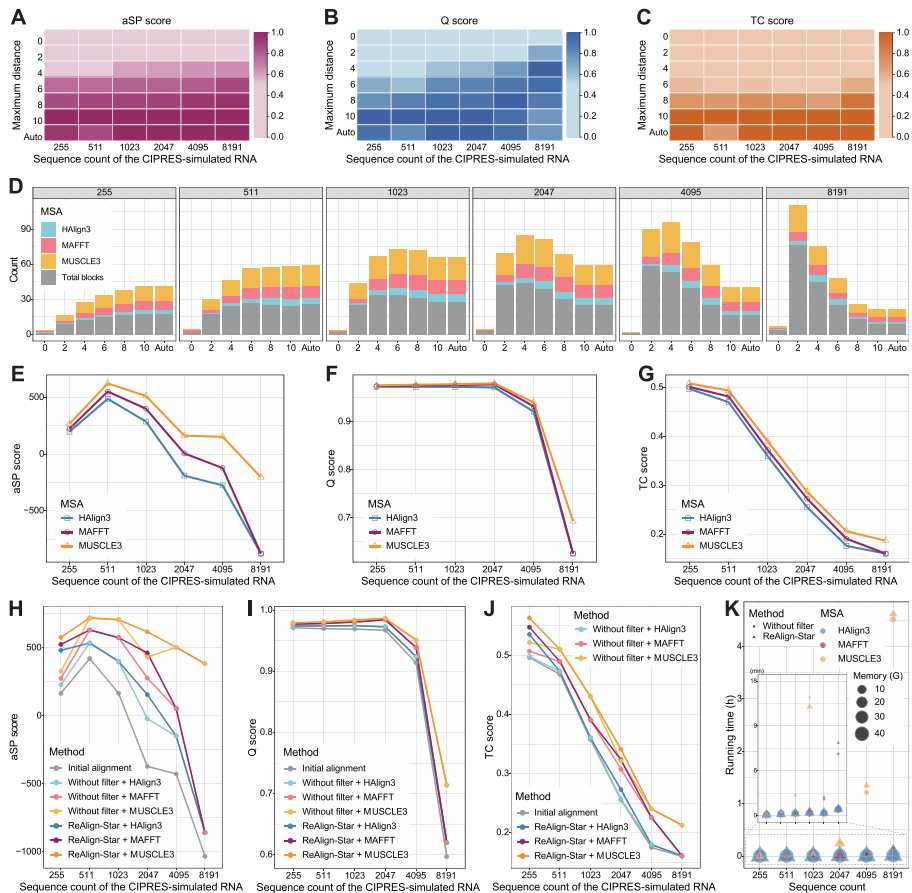


Fig. 3. Performance comparison based on the CIPRES-simulated rRNA datasets. **D** shows the average number of blocks identified across seven maximum distances under six different data sizes, along with the average number of blocks with improved SP scores after realignment using HAlign3, MAFFT, and MUSCLE3. The other information is the same as described in Fig. 2 legend.

In the 23S rRNA dataset (average similarity of 92%), MUSCLE3 maintained its lead over MAFFT and HAlign3 in realignment performance (Fig. 4B). In the SARS-CoV-2 genome dataset, with a high sequence similarity of around 99%, there was little need for local realignment when the number of sequences was small, leading to similar performances across all three tools (Fig. 4E). However, when the number of sequences is over 20 thousand, MUSCLE3's advantage diminishes, and HAlign3 begins to perform better in handling large datasets (Fig. 4H). Overall, when the number of sequences in a dataset is below 1,000, or the similarity is under 90%, MUSCLE3 offers the highest accuracy for realignment. But when sequence similarity is high, tool selection should depend on the dataset size: for smaller datasets, any tool works, while for datasets exceeding tens of thousands of sequences, HAlign3 is the optimal choice.

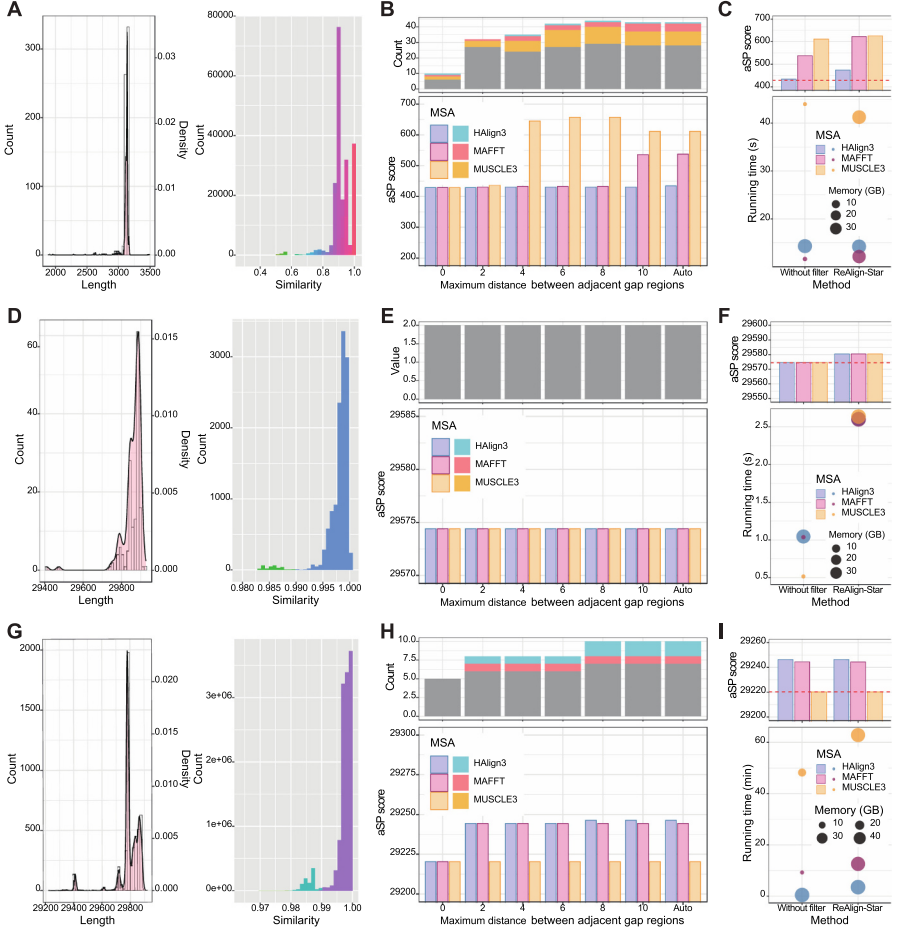


Fig. 4. Performance comparison based on real datasets. **A.** Histograms displaying the distribution of sequence lengths (left) and pairwise sequence similarity (right) in the 23S rRNA dataset. **B.** Average number of blocks and valid blocks identified across different maximum distances (top), and corresponding aSP scores after local realignment (bottom) for the 23S rRNA dataset. **C.** Comparison of aSP scores for ReAlign-Star with and without junk sequence filtering (top), along with the analysis of running time and memory usage for both methods (bottom). **D-I.** Corresponding analysis on the small (D-F) and large (G-I) SARS-CoV-2 genome datasets. While the histogram for the large SARS-CoV-2 genome is based on 5000 sequences from the origin dataset (G). The bubble chart illustrates the running time and memory consumption, with bubble height representing running time and bubble size indicating memory consumption.

3.3 Filtering and Realigning Junk Sequences is Crucial for Improving Accuracy in Low-Similarity Datasets.

We conducted ablation experiments to evaluate the effects of filtering out junk sequences and subsequently realigning them. Two methods were compared: the first, local realignment without filtering junk sequences (“Without filter”), and the second, ReAlign-Star,

which filters junk sequences before realigning and then aligns those sequences to the realigned profile. The runtime and peak memory usage were recorded using the script available at https://github.com/malabz/ReAlign-Star/blob/main/utils/time_mem.py.

In the mt-like genome dataset, when sequence similarity was below 90%, ReAlign-Star showed a clear improvement in aSP and Q scores over the without filter approach, though with slightly higher running time and memory usage. For similarities above 90%, the differences in aSP, Q, and TC scores were minimal, and the running time and memory usage of ReAlign-Star were nearly identical to those of “Without filter” (Fig. 2H-K). In the CIPRES simulated RNA dataset, ReAlign-Star led to significant improvements in aSP, Q, and TC scores, particularly in smaller datasets where junk sequence filtering had a more pronounced effect. Despite these gains, the running time and memory usage remained comparable to the “Without filter” approach (Fig. 3H-K).

In the 23S rRNA dataset, ReAlign-Star further improved aSP scores, although filtering junk sequences resulted in a slight increase in running time and memory usage (Fig. 4C). For the SARS-CoV-2 genome dataset, the impact of filtering junk sequences was most evident in smaller datasets (Fig. 4F), whereas in larger datasets (Fig. 4I), the effect was less pronounced.

4 Discussion

We developed ReAlign-Star, a realignment method tailored for star alignment tools. The approach first identifies and removes “junk sequences” and then performs local realignment on the remaining sequences. The junk sequences are then aligned to the realigned profile using sequence-to-profile alignment. Experiments were conducted on four simulated and three real datasets, and the results showed that ReAlign-Star consistently outperformed the initial alignment, with significant improvements in low-similarity datasets. Moving forward, we plan to further optimize the junk sequence filtering algorithm for large-scale datasets and explore other post-processing techniques to tackle complex tasks such as rapid genome alignment, continuously improving the practicality and performance of ReAlign-Star.

Acknowledgments. The work was supported by the National Natural Science Foundation of China (No. 62425107, No. 62450002) and the Municipal Government of Quzhou (No.2023D036).

References

1. Altschul, S.F.: Gap costs for multiple sequence alignment. *J. Theor. Biol.* **138**(3), 297–309 (1989)
2. Chao, J., Tang, F., Xu, L.: Developments in algorithms for sequence alignment: a review. *Biomolecules* **12**(4), 546 (2022)
3. Anson E.L., Myers E.W. (eds.) ReAligner: a program for refining DNA sequence multi-alignments. In: *Proceedings of the First Annual International Conference on Computational Molecular Biology* (1997)
4. Wallace, I.M., Higgins, D.G.: Evaluation of iterative alignment algorithms for multiple alignment. *Bioinformatics* **21**(8), 1408–1414 (2005)

5. Chakrabarti, S., Lanczycki, C.J., Panchenko, A.R., Przytycka, T.M., Thiessen, P.A., Bryant, S.H.: Refining multiple sequence alignments with conserved core regions. *Nucleic Acids Res.* **34**(9), 2598–2606 (2006)
6. Lyras, D.P., Metzler, D.: ReformAlign: improved multiple sequence alignments using a profile-based meta-alignment approach. *BMC Bioinf.* **15**, 1–18 (2014)
7. Manohar A., Batzoglou S. (eds.) TreeRefiner: a tool for refining a multiple alignment on a phylogenetic tree. In: 2005 IEEE Computational Systems Bioinformatics Conference (CSB'05), IEEE (2005)
8. Mokaddem, A., Hadj, A.B., Elloumi, M.: Refin-align: new refinement algorithm for multiple sequence alignment. *Informatica* **43**(4), (2019)
9. Zhan, Q., Fu, Y., Jiang, Q., Liu, B., Peng, J., Wang, Y.: SpliVert: a protein multiple sequence alignment refinement method based on splitting-splicing vertically. *Protein Pept. Lett.* **27**(4), 295–302 (2020)
10. Wei, Q., Zou, H., Zhong, C., Xu, J.: RPFam: a refiner towards curated-like multiple sequence alignments of the Pfam protein families. *J. Bioinform. Comput. Biol.* **20**(04), 2240002 (2022)
11. Thompson, J.D., Thierry, J.-C., Poch, O.: RASCAL: rapid scanning and correction of multiple sequence alignments. *Bioinformatics* **19**(9), 1155–1161 (2003)
12. Zhai, Y., Chao, J., Wang, Y., Zhang, P., Tang, F., Zou, Q.: TPMA: A two pointers meta-alignment tool to ensemble different multiple nucleic acid sequence alignments. *PLoS Comput. Biol.* **20**(4), e1011988 (2024)
13. Chen, J., Chao, J., Liu, H., Yang, F., Zou, Q., Tang F.: WMSA 2: a multiple DNA/RNA sequence alignment tool implemented with accurate progressive mode and a fast win-win mode combining the center star and progressive strategies. *Briefings Bioinf.* bbad190 (2023)
14. Zhang, P., Liu, H., Wei, Y., Zhai, Y., Tian, Q., Zou, Q.: FMAAlign2: a novel fast multiple nucleotide sequence alignment method for ultralong datasets. *Bioinformatics* **40**(1), btae014 (2024)
15. Tang F., Chao J., Wei Y., Yang F., Zhai Y., Xu L., et al.: HAlign 3: fast multiple alignment of ultra-large numbers of similar DNA/RNA sequences. *Mol. Biol. Evol.* **39**(8), msac166 (2022)
16. Katoh, K., Standley, D.M.: MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol. Biol. Evol.* **30**(4), 772–780 (2013)
17. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* **32**(5), 1792–1797 (2004)