Analysis of Greedy LCS-FIG Algorithm Performance

Mohammad Sadegh Sirjani

April 24, 2025

Abstract

This report presents a comprehensive analysis of the Greedy Algorithm for Longest Common Subsequence with Fixed-length Indel Gaps (LCS-FIG). We examine its performance characteristics, solution quality, and scalability through extensive experimental evaluation using DNA sequences of varying lengths and different gap constraints. The results demonstrate the algorithm's efficiency in terms of time and space complexity, while highlighting the trade-offs between execution speed and solution quality.

1 Introduction

The Longest Common Subsequence problem with Fixed-length Indel Gaps (LCS-FIG) extends the classical LCS problem by introducing constraints on the allowed gaps between matched elements. This variant has particular relevance in bioinformatics and text analysis applications where controlled spacing between matches is essential.

1.1 Algorithm Overview

The greedy approach to LCS-FIG provides a fast but non-optimal solution through the following steps:

1. Scan both sequences simultaneously
2. When matching characters are found, include them in the solution
3. Jump K+1 positions ahead in both sequences after a match
4. Move forward one position in the appropriate sequence when characters don't match

Complexity Analysis:

- Time Complexity: $\mathcal{O}(n + m)$
- Space Complexity: $\mathcal{O}(1)$

Algorithm 1 Greedy LCS-FIG Algorithm

Require: Sequences X[1..n], Y[1..m], fixed gap K
Ensure: Length of LCS-FIG

```
Initialize i ← 1, j ← 1, LCS length ← 0
while i ≤ n and j ≤ m do
    if X[i] = Y[j] then
        LCS length ← LCS length + 1
        i ← i + K + 1
        j ← j + K + 1
    else if X[i] ¡ Y[j] then
        i ← i + 1
    else
        j ← j + 1
    end if
end while
return LCS length
```

2 Experimental Setup

2.1 Test Parameters

- **Sequence Lengths**: [1000, 2000, 5000, 10000, 20000, 50000, 100000]
- **Gap Constraints (K)**: [2, 5, 10, 20, 50, 100, 200]
- **Number of Runs**: 10 runs per configuration
- **Sequence Type**: Random DNA sequences (A, C, G, T)

2.2 Evaluation Metrics

The algorithm's performance was evaluated using the following metrics:

1. Execution Time (seconds)
2. Solution Length (LCS length)
3. Processing Speed (characters/second)
4. Standard Deviations
5. LCS to Input Ratio

3 Results and Analysis

3.1 Time Performance

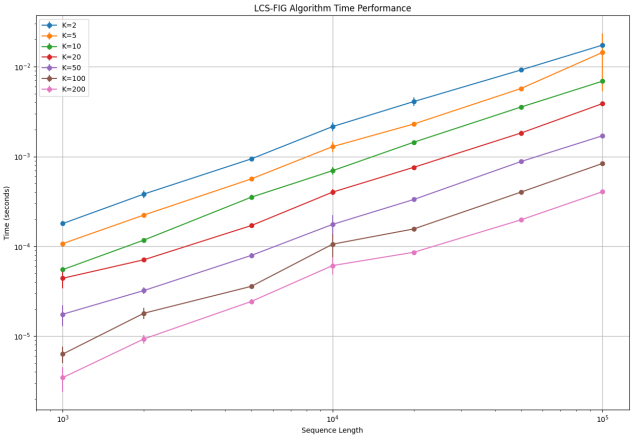

Figure 1: Time Performance Analysis

The time performance analysis (Figure 1) reveals several key characteristics:

- Linear growth in execution time with sequence length (log-log scale)
- Larger gap values (K) generally result in faster execution
- Consistent performance across multiple runs (small error bars)

3.2 Solution Quality

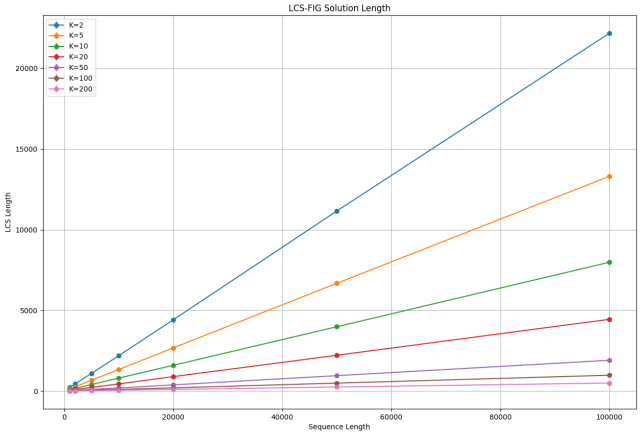

Figure 2: Solution Length Analysis

The solution quality analysis (Figure 2) shows:

- Sub-linear growth in LCS length with input size
- Inverse relationship between gap size and solution length
- Clear trade-off between execution speed and solution quality

3.3 Processing Efficiency

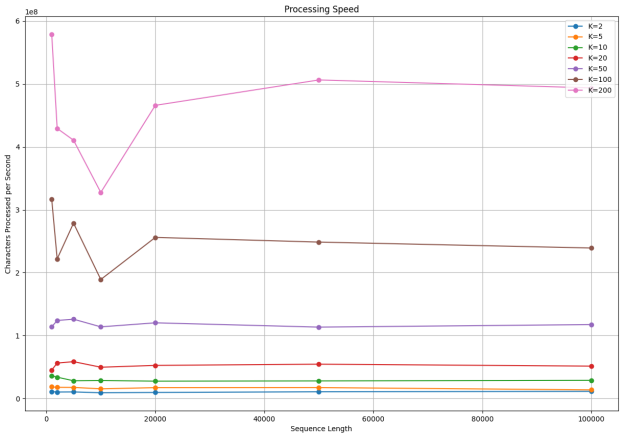

Figure 3: Processing Speed Analysis

The processing efficiency analysis (Figure 3) indicates:

- Relatively stable processing rates across sequence lengths
- Higher gap values achieve better throughput
- Slight performance degradation with very large sequences

4 Key Findings

4.1 Scalability

The algorithm demonstrates excellent scalability characteristics:

- Successfully processes sequences up to 100,000 characters
- Empirical results confirm theoretical linear time complexity
- Constant memory usage regardless of input size

4.2 Gap Constraint Impact

The gap parameter K significantly influences algorithm behavior:

- Larger K values improve processing speed
- Smaller K values produce longer (potentially better) solutions
- Optimal K value depends on specific application requirements

5 Trade-offs and Recommendations

5.1 Performance Trade-offs

1. **Speed vs. Quality**
   - Larger gap values increase speed but decrease solution quality
   - Smaller gap values provide better solutions but slower execution
2. **Memory vs. Optimality**
   - Constant memory usage achieved by sacrificing solution optimality
   - No backtracking or dynamic programming tables needed

5.2 Usage Recommendations

1. **Speed-Critical Applications**
   - Use larger gap values ($K \geq 50$)
   - Suitable for real-time processing of long sequences
2. **Quality-Critical Applications**
   - Use smaller gap values ($K \leq 20$)
   - Consider alternative algorithms if optimality is crucial
3. **Balanced Performance**
   - Use moderate gap values ($20 \leq K \leq 50$)
   - Provides good trade-off between speed and solution quality

6 Conclusion

The Greedy LCS-FIG algorithm proves to be an efficient solution for processing large sequences with fixed-gap constraints. Its linear time complexity and constant space usage make it particularly suitable for applications where speed and memory efficiency are prioritized over solution optimality.

The experimental results demonstrate that the algorithm can effectively process sequences of 100,000+ characters while maintaining consistent performance characteristics. The gap constraint parameter provides a flexible means of tuning the algorithm's behavior to meet specific application requirements.

7 Future Work

Several directions for future research include:

1. Comprehensive comparison with other LCS-FIG algorithms
2. Analysis on different types of sequence data
3. Investigation of parallel processing opportunities
4. Development of adaptive gap constraint strategies