Global Alignment
Algorithm
Optimization

COEN 279 - Design and Analysis of Algorithms
Santa Clara University

Kurt Ruegg
Sui Fung Alex Wong
Table of Contents

1. Abstract
2. Introduction
   Objective
   What is the problem?
   Why this is a project related to this class?
   Why other approach is no good? And why you think your approach is better?
   Statement of the problem
   Area or scope of investigation
3. Theoretical bases and literature review
   Definition of the problem
   Theoretical background
   Related research to solve the problem
   Advantage and Disadvantage of the above mentioned research
   Our solution to solve this problem
   Where our solution different from others
   Why is our solution better
4. Hypothesis
5. Methodology
   How to generate/collect input data
   How to solve the problem
   Algorithm Design
   Language used
   Tools used
   How to generate output
6. Implementation
   Design
7. Data Analysis and Discussion
   Output Generation
   Output Analysis
   Abnormal Cases
8. Conclusion
   Summary and Conclusion
   Recommendation for Future Studies
9. Bibliography
10. Appendices
    Program Source Code, Documentation and Output files
1. Abstract

The Needleman-Wunsch algorithm was the first computer algorithm developed to aid in aligning DNA sequences. Aligning DNA sequences optimally is an essential part of the field of bioinformatics. With a pair of aligned DNA sequences, researchers can determine where the location of mutations, how closely related two organisms are, and a number of other things. The Needleman-Wunsch algorithm is a very basic dynamic programming approach to solving this problem and there is room to improve both the asymptotic space and time complexity and to improve the constant factors. We will use some optimizations for LCS (longest common subsequence) problems from existing literature to improve the algorithm as well as develop a "pruning" method to improve the constant factor for time complexity.
2. **Introduction**

**Objective**
The objective of this project is to create a new sequence alignment methods, by analysing different existing methods, including both exact methods and approximated methods. We aim to come up with a new sequence alignment method by pruning that improves the running time.

**What is the problem?**
The problem is that when the compared sequences become huge, the table size will become bigger as it is corresponding to the product of the sequence lengths, and the algorithm needs to compute all of the entries of the table. Part of the table contains low scores that will never used, so we suggest a solution where the algorithm could avoid computing all of the entries of the table by pruning.

**Why this is a project related to this class?**
This project is mainly about sequence alignment method, which is related to longest common sequences. There are many different algorithms that can provide exact or approximate results with different space complexity and time complexity. Lots of techniques and ideas can be learned by analysing them. Therefore this project is suited for the class.

**Why other approach is no good? And why you think your approach is better?**
The other approaches that we came across are focusing on different parts of the problem. They are either focusing on the score, or both time and space complexity, while neglecting the other parts, such as neglecting space complexity, or the score is far off from the optimal score. In order to provide close result to the Needleman-Wunsch algorithm, our approach will be focusing on reducing the running time.

**Statement of the problem**
The problem is that when the compared sequences become huge, the table size will become bigger as it is corresponding to the product of the sequence lengths, and the algorithm needs to compute all of the entries of the table. Part of the table contains low scores that will never used, and those entries can be avoided to compute.

**Area or scope of investigation**
Sequence Alignment Methods
Challenges in optimizing the result while trying to reduce the running time.
3. Theoretical bases and literature review

Definition of the problem
To create an algorithm that provides approximated alignment scores that are closer to the optimal alignment scores from Needleman-Wunsch algorithm, by using the pruning technique that reduce the entries of the table to be computed. To achieve closer score as much as possible, while reducing the overall running time.

Theoretical background
The Needleman-Wunsch is based on the Longest Common Subsequence (LCS) dynamic programming algorithm. The major difference is that the Needleman-Wunsch algorithm introduces scores for sequence matches, mismatches, and gaps in the sequence. Because this algorithm was created with the goal of aligning DNA sequences, the theoretical foundation of this algorithm takes genetics into consideration. First, the use of the gap score allows the algorithm to account for the possibility of insertions or deletions in the genetic sequence, which are both common occurrences in natural DNA. Second, the scores for each of the three possible alignments can be adjusted and fine-tuned to get the alignment criteria most useful for biological research use.

Related research to solve the problem
Below are a few of the research that we come across:

“Fast Global Sequence Alignment Technique”
In this paper a new alignment technique is proposed. It runs both sequences at the same time, then create barriers to breakdown the sequences into multiple subsequences, and do subsequence matching.

“Hirschberg’s Algorithm”
In this paper, a version of the LCS which only uses linear time \(O(n + m)\), where \(n\) and \(m\) are the size of the strings is proposed. This algorithm uses a constant factor more time than the basic Needleman-Wunsch algorithm, but improves the asymptotic space complexity.

“Fast Sequence Alignment Algorithm Using Bloom Filter”
This paper discusses a method to bloom filters to approximate the which sequence in a database of sequences best aligns with a given DNA sequence. This method is not exact, as Bloom Filters only approximate answers, but the results show that the method was able to come close to the quality of exact algorithms, while taking up considerably less space and time.
Advantage and Disadvantage of the above mentioned research

The Hirschberg Algorithm is part of the natural improvement of global alignment algorithms and will most likely be implemented are part of our algorithm.

Our solution to solve this problem

With our proposed algorithm, it will be able to avoid computing some of the entries in the table of Needleman-Wunsch algorithm, as their parent score will be beyond the threshold that they are unlikely to be chosen to the optimal result.

In more detail, our algorithm will make an initial “guess.” We will then build the table and whenever a row reaches a value that cannot possibly beat our initial guess, we will prune the rest of the row. For example, looking at the table below and assuming an initial guess of 0: when building the second row, we could prune after reaching the cell that has -3 because even if the rest of the string matches, we would not beat or our initial guess. Therefore, it is a waste of resources to continue computing.

Making a guess could be done by either:

1. Creating a heuristic algorithm that quickly runs and gives an answer that is most likely not optimal.
2. Arbitrarily picking a minimum score. With this method, there is a chance that the entire table would be pruned, meaning there is no way to reach this score with our two sequences. In this case, the minimum score would need to be relaxed (increased). With this method, we would be fishing for an optimum guess that would prune a maximal amount of the tree, while returning a valid answer.

Where our solution different from others
Our solution will be based on Needleman-Wunsch algorithm, and by adding pruning to the algorithm, the score will be closer to the original algorithm and having reduced computation time. As opposed to others that they are trying to come up with a new algorithm completely and achieve a closer score instead.

Why is our solution better
Our solution is better because we aimed to maintain the optimal scores from Needleman-Wunsch algorithm, while pruning the suboptimal scores that are beyond the threshold.
4. **Hypothesis**

We hypothesize that we will be able to combine a linear space sequence matching algorithm (the Needleman-Wunsch algorithm augmented by Hirschberg’s space efficient alterations) with our pruning strategy to produce an algorithm that is both space efficient and fast. Of the two pruning strategies, we suspect that the heuristic method will be a more efficient and offer a more elegant solution.
5. Methodology

How to generate/collect input data
Shorter input sequences will be randomly generated, as they are used to test the functionality of the program. Longer input sequences will be taken from various DNA databases through the Internet, such as DNA Data Bank of Japan, as they are used to test the time complexity of the program.

How to solve the problem
In order to solve the problem, first we need to implement the Needleman-Wunsch algorithm, as it can provide the optimal solution to the alignment with time complexity $O(mn)$. We will then add on the basic space optimizations that we have found during our literature review. Since the table size of Needleman-Wunsch algorithm is the product of the sequence lengths, we could try to avoid computing the complete table by pruning. Hence, as a last step, we need to implement some pruning methods and test around how to improve the time complexity and space complexity.

Algorithm Design

Needleman-Wunsch algorithm with Pruning

- Define min_score
  
  $\text{min}\_\text{score} = \text{gap} \times \text{abs}(x - y) + (\text{mismatch} \times 3 + \text{match}) \times \text{min}(x, y) / 4$

- When running through the matrix, add an additional clause to skip the rest of the row if it is below the min_score.
  
  $\text{if } (i < j \&\& (y - j) \times \text{match} + \text{matrix}[i-1][j-1] < \text{min}\_\text{score}) \{$
  
  $\text{matrix}[i][j] = \text{INT}\_\text{MIN};$
  
  $j = y;$

- During traceback, we also need to make sure our path is not getting cut off by the pruned entries. If it is getting cut off, indicates failure. (But this should never happen.)
  
  $\text{if } (s1\_\text{pos} > 0 \&\& s2\_\text{pos} > 0$
  
  $\&\& (\text{matrix}[s1\_\text{pos}-1][s2\_\text{pos}-1] == \text{INT}\_\text{MIN}$
  
  $|| \text{matrix}[s1\_\text{pos}-1][s2\_\text{pos}] == \text{INT}\_\text{MIN}$
  
  $|| \text{matrix}[s1\_\text{pos}][s2\_\text{pos}-1] == \text{INT}\_\text{MIN})) \{$
  
  "$\text{success} = \text{false;}
  
  \text{return } -1;$
  
}
**Needleman-Wunsch algorithm with Barriers**

- Find the barriers, they are the index $i$ such that $seq1[i] == seq2[i]$.
- Apply Needleman-Wunsch algorithm on the subsequences between the barriers.
- Finally adding the scores and concatenate the subsequences together for the final result.

**Language used**

We used C and C++ for this project.

**Tools used**

Terminal
gcc/g++
Makefile
Xcode
Any Text Editor

**How to generate output**

We will use the same score setup for various algorithms, such as our own algorithm and Needleman-Wunsch algorithm. The program will be able to log the running time for each input sequence pair with different algorithms, then we can compare the running time and the score between the algorithms.
6. Implementation

Design

Framework

main

readFiles and storeData

doAlgorithm

- record starting time $s$
- pick an algorithm and run
- record ending time $t$
- print result
  - running time $(t - s)$
  - alignment score
  - aligned sequences (if they are short)

AlignMaster

- data
  - sequence 1
  - sequence 2
  - scoring system

- algorithms
  - Needleman-Wunsch
  - Hirschberg
  - Needleman with Barrier
  - Needleman with Pruning

- read the files and store the input sequences and the scoring system
- run each implemented algorithm with the stored data and print the result
Needleman-Wunsch algorithm with Pruning

- establish a minimum acceptable score value
- run through Needleman algorithm, pruning a cell and the rest of the row when we cannot possibly reach our minimum score with particular cell in path
- reduces the running time needed to calculate unnecessary cells

Needleman-Wunsch algorithm with Barriers

- basic idea is to split the original sequences into subsequences, so the matrix size can be reduced
- optimal path might be off the sub-matrices, resulting this algorithm to have a really bad score
- significantly reduced the space requirement and running time
- if no barriers exist, then it is just a normal needleman
7. **Data Analysis and Discussion**

**Output Generation**

One of the output files: `output/13-12-2`

```bash
$ ./alignmaster seq/13.dat seq/12.dat scoring/2
```

**Settings:**

- seq1.length: 822
- seq2.length: 852
- match: 2
- mismatch: -1
- gap: -2

**Needleman:**

- Running Time: 30248 usec
- Score: 153

**Hirschberg:**

- Running Time: 90190 usec
- Score: 153

**Needleman with Barrier:**

- Running Time: 634 usec
- Score: -162

**Needleman with Pruning:**

- Running Time: 26934 usec
- Score: 153
Output Analysis

**Hirschberg’s algorithm**
The Hirschberg’s algorithm performs slower than the Needleman-Wunsch algorithm, but does have a lower space complexity. It also produces the same quality of answer as the Needleman-Wunsch algorithm. This is what we expected because the Hirschberg algorithm requires recursively splitting the problem and requires more comparisons and rewrites to save space and ensure accuracy.

**Needleman-Wunsch algorithm with Barriers**
This variation of Needleman-Wunsch algorithm runs a lot faster and uses much less space. The problem is that getting local optimal solution does not guarantee global optimal solution. When the difference of the sequence lengths is huge, this algorithm does surprising well that the score is close to the optimal. On the other hand, when the difference is minimal, the score is too far off and it cannot be used in practical. If barriers can be chosen properly, this algorithm might be able to produce better result.

**Abnormal Cases**

**Pruning**
Pruning is implemented by establishing a min_score, which is the minimum score that we will accept. If any cell cannot possibly be part of a path that reaches our min_score, the rest of the row is pruned. This method works best when we select our min_score to be equal to the actual score. However, the worst case is when our min_score is slightly larger than our actual score and requires $O(nm)$ time to determine that we have selected a poor min_score. This creates an interesting challenge because it makes probing for a near ideal min_score very risky.

To explain in more detail using an example:
In the figure below, the ideal ideal min_score is 0 for the given sequences and scoring system. The ideal is 0 because it exactly matches our actual score. However, we run into a major problem if we choose 1 as our minimum score. In fact, choosing this value results in the worst possible case.

**Min_score = 0 (left figure):** we have the pruning results as shown on the left hand side. We are able to solve the problem because no cells in the path are pruned. Also, we have pruned the theoretical maximum number of cells for this problem.

**Min_score = 1 (right figure):** when we choose a min_score that is 1 above our actual score, the result is our theoretical worst case. Not only do we not find a correct answer for the full path, but we also calculate a large number of cells in the matrix. If, for example, we selected a very high min_score, such as 6, we would be able to prune away the entire table. We would still not
find a correct answer, but at least we wouldn’t waste time calculating a large number of cells. This is not the case when we choose a value slightly above our actual score.

**Who Wants to be a Millionaire Effect:** increasing our min_score will always prune more of the table, thereby improving speed. However, once we cross a threshold, we lose all of our gains because our answer is no longer correct.

This creates the challenge of selecting a good value to prune. If the value is too low, very few cells in the table are pruned, but we are almost guaranteed to find the correct answer. Picking a larger value increases the number of cells pruned, but also increases the chance that we will produce an incorrect answer.

In order to choose an effective value we used:

\[
\text{min\_score} = \text{gap} \times \text{abs}(x - y) + (\text{mismatch} \times 2 + \text{match}) \times \text{min}(x, y) / 4
\]

The above equation reflects a conservative choice that finds the score if the sequence is composed of \(\frac{1}{4}\) matches, \(\frac{1}{2}\) mismatches, and enough gaps to fill in the rest of the difference between the lengths of the subsequences. This choice produces a very modest speed increase for all lengths that we tested and does not produce any failures. A more aggressive min_score could produce a much better speed increase.
8. Conclusion

Summary and Conclusion

Based on the output that we gathered, the Needleman-Wunsch algorithm with Pruning is having the same time complexity and space complexity as the original Needleman-Wunsch algorithm. It rarely achieves a better speed if the min_score fits the given input sequences. Only for very specific instances does the pruning method give substantial time improvement. The current version is not effective for the general case and is not viable for practical use.

For the Needleman-Wunsch algorithm with barriers, the running time is significantly faster, and the used space is significantly smaller. But the downside is that the barriers enforced the matrix to traceback the barriers, which is highly possible to be a suboptimal solution. Interestingly, if the difference of the input sequence lengths is huge, this algorithm provides a close solution to the optimal one, with much faster running time.

Recommendation for Future Studies

Needleman-Wunsch algorithm with Pruning Improvement:
Improving the min_score is essential for making the pruning algorithm viable. This would require an understanding of the input sequences. One way to develop a better min_score would be to run the pruned Needleman algorithm on a large number of combinations of sequences from a database of DNA sequences. Using the distribution of scores from this testing could yield a min_score that would give useful speed improvements. For example, we could find the mean and standard deviation of the scores and choose a min_score that is larger than the actual score in nearly all cases, but still reasonably close to the actual score, so as to prune large chunks of the table.
9. Bibliography


DNA Data Bank of Japan, http://www.ddbj.nig.ac.jp/
10. Appendices

Program Source Code, Documentation and Output files

They can be found on the following Github repo:
https://github.com/kruegg21/coen279-project