American University of Armenia

Zaven & Sonia Akian College of Science and Engineering

CS 245, Bioinformatics Course Project

K-Band DP for Pairwise Comparison

Sona Bezirganyan, Lusine Davtyan

Spring 2022

Contents

Chapter 1

Introduction

1.1 Problem Setting and Background

Sequence alignment is a procedure in Bioinformatics for comparing two or more sequences and finding regions of similarity by arranging the sequences in DNA, RNA or protein level. Sequence alignment is widely used for analysing structural, functional and evolutionary relationships between the sequences being compared. There are three types of alignments—global, local and semi-global [Mou04]. In terms of the number of sequences being compared, the sequence alignments can be pairwise sequence alignments and multiple sequence alignments. In the scope of this project, we will consider pairwise sequence alignments. Pairwise sequence alignment is the sequence alignment between two sequences. The problem of finding optimal alignments is mainly solved by dynamic programming. Modifications of the basic algorithm of finding optimal alignments between the sequences allow to improve the time complexity in case of the sequences having certain characteristics. Further, we will discuss one of those modifications of the basic algorithm for similar sequences.

1.2 The Structure of the Report

The project paper consists of 3 sections. The second section is devoted to introducing K-band algorithm for pairwise sequence alignment, its complexity analysis and our implementation. The third section discusses the comparison of our implementation with the existing package, conclusions and possible directions for the future work.

Chapter 2

The Algorithm

2.1 Comparing Similar Sequences

Two sequences are called similar if the scores of the optimal alignments are very close to the maximum possible score. We will assume that the two sequences have the same length. And if we construct the dynamic programming table, by following the path through the diagonal, we will get a unique alignment that does not have to be the optimal one. If it is not the optimal one, we need to insert spaces and, as a result, divert from the main diagonal, for finding a better alignment. And as we have similar sequences we will have the paths of our optimal alignments near the main diagonal. Hence, there is no need for us to compute the entire DP table. It is sufficient to consider a narrow band around the main diagonal. And, to this end, we should decide on the width of the band as it may change depending on the given sequences. For finding out the width of that band, K-band algorithm was proposed.

2.2 K-band algorithm

The idea behind the K-band algorithm is considering a narrow strip around the main diagonal of the DP matrix for finding optimal alignments between similar sequences. The description of the K-band algorithm and the complexity analysis are based on the book “Bioinformatics: Sequence and Genome Analysis” [SM97]. Let us denote the DP matrix for aligning two sequences of same length by a. K-band algorithm computes the entries of the matrix in a band around the main diagonal having width $2k + 1$. After filling in the entries inside the band, the entry `a[n,n]` will contain the optimal alignment considering only the part of the table inside the band. The time complexity of the algorithm will be $O(kn)$ instead of

$O(n^2)$ of the basic algorithm. And when k is much smaller than n, the running time of this algorithm will be much efficient. In order to understand which entries we should fill in, we should check if the current position `(i,j)` is inside the strip with the criterion:

$$InsideStrip(i, j, k) = -k \leq i - j \leq k$$

In the basic algorithm, for filling in a current entry `a[i, j]` we should consult the entries `a[i-1, j-1]`, `a[i, j-1]`, `a[i-1, j]`. In the K-band algorithm, we should beforehand check if the positions `a[i, j-1]` and `a[i-1, j]` are inside the band as they can go beyond the band. There is no need to check if the entry `a[i-1, j-1]` is inside the strip because it is in the same diagonal as the current entry, so it is inside the strip. We first choose a value of k, and proceed with the algorithm to update the initial value if needed and find the right one. If `a[n, n]` is greater than or equal to the best possible score obtained from aligning the sequences having $k+1$ or more space pairs inserted, we stop as we have found the right value for k. Otherwise, we double the k and repeat the check. The best possible score from aligning with at least $k+1$ space pairs is given below which is calculated assuming that there are $k+1$ space pairs and the other pairs match:

$$M(n-k-1) + 2(k+1)g$$

where M is the match score which is positive, and g is the gap penalty which is non-positive. So, the stopping condition is

$$a_k[n,n] \geq M(n-k-1) + 2(k+1)g.$$

2.3 Complexity Analysis

If we initialize the value of k to be 1, we will have the value of each updated k to be a power of two since we double the k when it does not satisfy the stopping condition. And the total complexity when running the algorithm many times will be

$$n + 2n + 4n + ... + kn \leq 2kn.$$

We can have an upper bound on k assuming we haven't stopped before:

$$\frac{k}{2} < \frac{M_n - a_{\frac{k}{2}}[n,n]}{(M-2g)} - 1.$$

When $a_k[n,n] = a_{\frac{k}{2}}[n,n]$, we will get

$$k < 2 \cdot \left(\frac{Mn - sim(s,t)}{M - 2g} - 1 \right)$$

where `sim(s,t)` is $a_k[n,n]$—the optimal score. Otherwise, if $a_k[n,n] > a_{k/2}[n,n]$ we have more than $\frac{k}{2}$ space pairs in the optimal alignments, resulting in the following inequality:

$$sim(s,t) \leq M\left(n - \frac{k}{2} - 1\right) + 2\left(\frac{k}{2} + 1\right)g$$

and for having an upper bound on k we can equivalently rewrite the above equation as

$$k \leq 2 \cdot \left(\frac{Mn - sim(s,t)}{M - 2g} - 1\right)$$

$(M - 2g)$ is a constant, implying that the time complexity will be $O(dn)$ where d is the difference between Mn and `sim(s,t)` which are the best possible score and the optimal score respectively. So, with higher similarity between the two sequences, we will get the answer faster.

2.4 Implementation of the Algorithm

The design of the algorithm implementation is quite similar to the sequence of steps mentioned in the algorithm description. We first determine the value of k by starting from $k = 2$ and iteratively updating it until the stopping condition is satisfied. When k is determined, the DP matrix is already constructed since it was needed to perform the stopping condition check. It is important to observe that depending on the type of the alignment, the initialization of the first row and column differs. Unlike global alignment, where we use the gap penalties for this, in the case of local and semiglobal alignments, we initialize the first row and column with 0-s. Next, we compute the score depending on the type of the alignment—global, local, and semiglobal. Finally, we perform traceback to find all optimal alignments between the given 2 sequences. If the problem is to find the global alignments, we start the traceback from `a[n][n]`. If the alignment is local, we traceback from all positions containing a maximal value. Otherwise, if we have semiglobal alignment, the traceback is performed from the maximum entries in the last row and column of the DP matrix. Figure 2.1 shows the flowchart of the described algorithm. The program first outputs the optimal alignment found, then the K-band DP matrix, and finally the score of an optimal alignment.

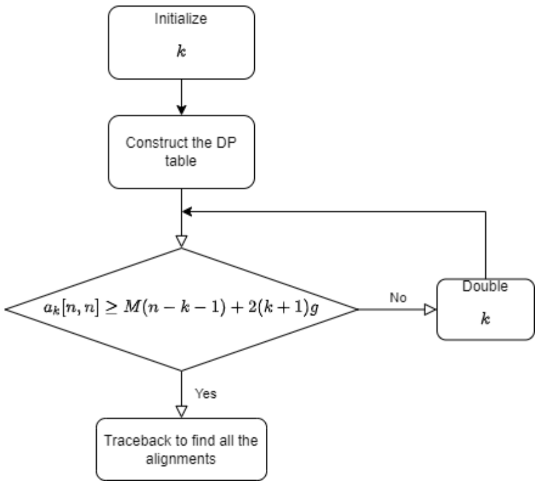
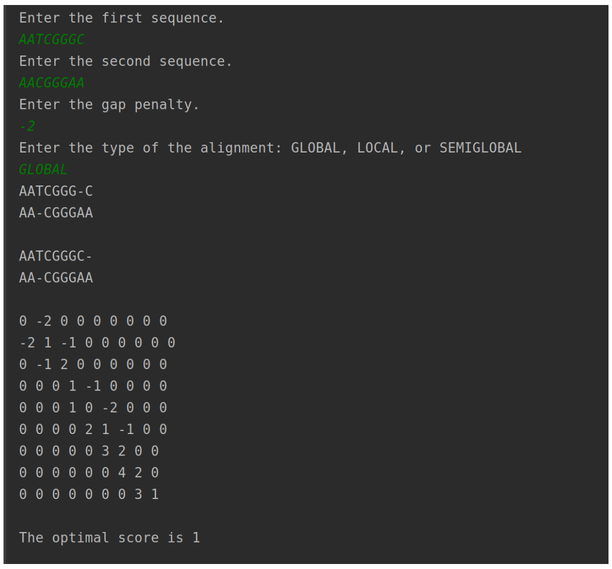

Initialize
k
Construct the DP table
$a_k[n,n] \geq M(n-k-1)+2(k+1)g$
No
Double
k
Yes
Traceback to find all the alignments

Figure 2.1: Flowchart of the K-band algorithm

2.5 Implementation Details

We used `C++` to implement the algorithm. A sample output is shown in Figure 2.2. To execute the algorithm, it is enough to compile and run the `k_band_dp.cpp` file, and then provide the requested information—the sequencces, the gap penalty, and the type of the alignment.



```
Enter the first sequence.
AATCGGGC
Enter the second sequence.
AACGGGAA
Enter the gap penalty.
-2
Enter the type of the alignment: GLOBAL, LOCAL, or SEMIGLOBAL
GLOBAL
AATCGGG-C
AA-CGGGAA

AATCGGGC-
AA-CGGGAA

0 -2 0 0 0 0 0 0 0
-2 1 -1 0 0 0 0 0 0
0 -1 2 0 0 0 0 0 0
0 0 0 1 -1 0 0 0 0
0 0 0 1 0 -2 0 0 0
0 0 0 0 2 1 -1 0 0
0 0 0 0 0 3 2 0 0
0 0 0 0 0 0 4 2 0
0 0 0 0 0 0 0 3 1

The optimal score is 1
```

Figure 2.2: Sample input and output of the program

Chapter 3

Comparison and Conclusions

3.1 Comparing with another package

We have compared the results of our implementation with the results of `BioPython` [Coc+09]—a collection of tools for biological computations written in Python. To that end, we have generated 10000 random sequences and compared the resulting alignments. As a result, in the case of comparing global alignments, for example, 9998 out of 10000 sequences were identical. In the rest 2 pairs, our implementation found a subset of the optimal alignments. One of those pairs is `s=GCCGGTCGGA` and `t=GCCGGTCGGA`. The reason behind this is that when determining the value of k, the K-band algorithm stops as soon as it makes sure that at least one optimal alignment can be found. Thus, alignments that would "spread" outside of the k strip are left out in our case.

3.2 Conclusions

We discussed a modification of the basic DP algorithm to align 2 sequences. Instead of filling the whole DP matrix, it fills a K-band near the main diagonal of the matrix. As a result, if the sequences are similar, we have a significant improvement in the time complexity—$O(dn)$ instead of $O(n^2)$ of the basic algorithm. However, we do have trade-offs here. As discussed in the previous section, the K-band DP may not find all of the optimal alignments, since some of them may be left out of the k strip.

3.3 Future Work

In the future, the implementation of this algorithm can be extended to align sequences of different lengths. Also, space efficient algorithms should be considered for working with long sequences, as in the real world setting, the genetic sequences to be aligned are very long and need to be handled efficiently.

Bibliography

[SM97] J.C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Pub., 1997.

[Mou04] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Series. Cold Spring Harbor Laboratory Press, 2004.

[Coc+09] Peter JA Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (2009), pp. 1422–1423.