Sequence Alignment in DNA/RNA

Final Report

PREPARED BY

Joshi Niranjan Suhas (160001026)

Kanishkar J (160001028)

Acknowledgement

We are highly indebted to Dr Kapil Ahuja, Associate Professor, IIT Indore for his guidance and constant supervision as well as for providing an opportunity to work on Algorithm analysis in real life. We have gained a lot of knowledge through this project, and have had our first experience on a research project.

Overview

Sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences.

If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as insertion or deletion mutations introduced in one or both lineages in the time since they diverged from one another.

Here we implemented two algorithms to solve the problem of sequence alignment :

1. Smith - Waterman Algorithm
2. Progressive alignment construction for multiple sequence alignment.

Smith-Waterman Algorithm

Smith-Waterman Algorithm is used for local alignment of two sequences. It gives the best possible alignment of two sequences based on substitution matrix and gap-scoring scheme. Dynamic Programming is used to solve the recurrence relation in bottom up approach. This is done by filling the scoring matrix. Given here is an example of matching of two sequences of DNA

Sequence 1: **TGCGGGCCCGCTA**

Sequence 2: **TAGCCCTA**

Here we find that there is mismatch between **G-A**. The gaps are also created while matching to create a perfect match. For each match there is a positive score and for every mismatch there is a negative score. Creation of every gap requires certain gap penalty, this is also negative.

```
TGCGGGCCCGCTA
||      |||     |||
TA____GCC___CTA
```

Recurrence Relation of Smith's Algorithm

Let S(n,m) denote the maximum score for DNA sequences A and B of length n and m respectively. S(n,m) is the maximum of:

1. If A_n=B_n then S(n-1,m-1)+match_score, else S(n-1,m-1)+mismatch_score. Let this be denoted by k.
2. Modifications due to indels (insertions and deletions) i.e.

S(n,m-1)+gap_penalty and S(n-1,m)+gap_penalty.

Hence,

S(n,m)=max(k, S(n-1,m)+gap_penalty, S(n,m-1)+gap_penalty, 0)

C++ Implementation for Smith-Waterman Algorithm

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;

void smith_waterman(string &a,string &b,ll s,ll mm,ll g){
//Dynamic Programming Function for finding alignment

    ll n,m,max_score=0,iidx=0,jidx=0;
    n=a.length();
    m=b.length();
    ll dp[n+1][m+1];
    int bt[n+1][m+1];        //Declaration of matrix

    for(ll i=0;i<=n;i++){
        dp[i][0]=0;
        bt[i][0]=0;           //Initialisation
    }
    for(ll j=0;j<=m;j++){
        dp[0][j]=0;
        bt[0][j]=0;
    }

    for(ll i=1;i<=n;i++){       //Computation
        for(ll j=1;j<=m;j++){
            (a[i-1]==b[j-1]) ? dp[i][j]=dp[i-1][j-1]+s :
dp[i][j]=dp[i-1][j-1]+mm;

dp[i][j]=max(max(dp[i][j],(ll)0),max(dp[i-1][j]+g,dp[i][j-1]+g));
```

```
            if(dp[i][j]==dp[i-1][j-1]+s && a[i-1]==b[j-1]) bt[i][j]=2;
            else if(dp[i][j]==dp[i-1][j]+g) bt[i][j]=3;
            else if(dp[i][j]==dp[i][j-1]+g) bt[i][j]=1;
            else bt[i][j]=2;

            if(max_score<dp[i][j]){
                max_score=dp[i][j];
                iidx=i;
                jidx=j;
            }

        }
    }


    cout<<"\nScore: "<<max_score<<endl;

    string ans1="",ans2="";
    ll i=iidx,j=jidx;
    while(bt[i][j]>0){
        if(bt[i][j]==2){
            ans1=a[i-1]+ans1;
            ans2=b[j-1]+ans2;
            i--;j--;
        }
        else if(bt[i][j]==1){
            ans1="_"+ans1;
            ans2=b[j-1]+ans2;
            j--;
        }
        else if(bt[i][j]==3){
            ans1=a[i-1]+ans1;
            ans2="_"+ans2;
            i--;
        }
    }
    cout<<"\n Alignment:\n"<<ans1<<'\n'<<ans2<<endl;
}

int main(){
```

```
    string a,b;             //Declaration of Variables
    ll scoring;
    ll gap_penalty,mismatch;

    cout<<"\n--- Program for DNA/RNA Sequence Alignment ---\n";
    cout<<"\n Match Score: ";          //Input
    cin>>scoring;
    cout<<"\n Mismatch Score: ";
    cin>>mismatch;
    cout<<"\nGap Penalty: ";
    cin>>gap_penalty;
    cout<<"\nSequence 1: ";
    cin>>a;
    cout<<"\nSequence 2: ";
    cin>>b;

    smith_waterman(a,b,scoring,gap_penalty,mismatch);       //Alignment

    return 0;
}
```

Complexity Analysis of Smith Waterman Algorithm

The algorithm takes two sequences of length n and m. Bottom up approach is used. Two matrices of size (n+1) x (m+1) is used for storing values of the scores and backtracking. The algorithm terminates when all the matrix is filled. Following is the complexity required at each step:

1. Initialization of the matrix - Two for loops initialize first row and first column with 0. Complexity: O(m+n)
2. Bottom up filling of scoring matrix - The matrix for the scores is filled by using two for loops in O(mn) computational steps. At the same time the backtracking matrix is also filled based on the position of the maximum element chosen.

3. Filling of backtracking matrix:
    a. If maximum is left then value 1 is assigned.
    b. If maximum is up-left then value 2 is assigned.
    c. If maximum is up then value 3 is assigned.
4. Backtracking is done from the highest score in the scoring matrix till zero is encountered. Complexity: O(mn).

Thus the overall complexity of the algorithm is O(nm).

Drawbacks of Smith's Algorithm

1. The Smith Waterman Algorithm has space complexity of O(nm). Thus for large length of sequences (which is usually case with DNA/RNA) huge amount of memory is required.
2. Smith's Algorithm cannot be used for Multiple Sequence Alignment as it is not efficient in terms of time and space.
3. Multiple Sequence Alignment using above algorithm is an NP-complete problem. Consider n sequences of length m each. As two sequences require two dimensional matrix, n sequences require n-dimensional matrix with each dimension of length m. Therefore the space and time complexity is O(m^n). Therefore, as number of sequences increases, the complexity of search space increases exponentially. For the very same reason, Progressive Algorithm which is not as accurate as Smith's Algorithm is used. It uses greedy approach and solves problem more efficiently.

Progressive Alignment Construction

As We have seen earlier, the Smith - Waterman algorithm has a complexity of O(mn) where m is the length of the first sequence and n the length of the second sequence. Now in real life each DNA sequence can be as long as 2 million base pairs. The Human DNA sequence has a length of 27,000 base pairs. And comparing around 100 DNA sequences is computationally impractical. Hence We use a heuristics algorithm. The most popular heuristic approach to this class of problems is known as known as progressive technique.

Approach

This algorithm is implemented using Greedy algorithm design approach. The algorithm is very large and we couldn't implement the It. Hence we will explain its implementation.

Implementation

- We build up a final Multi sequence alignment by combining pairwise alignments beginning with the most similar pair and progressing to the most distantly related.
- All progressive alignment methods require two stages: a first stage in which the relationships between the sequences are represented as a tree, called a guide tree, and a second step in which the MSA is built by adding the sequences sequentially to the growing MSA according to the guide tree.

- The initial guide tree is determined by an efficient clustering method such as neighbor-joining or UPGMA, and may use distances based on the number of identical two letter sub-sequences

Procedure

1. Using standard pairwise alignment, calculate a matrix of distances (alignment scores) between each pair of sequences. Consider this as an N-clique G, where edge {i,j} is labeled with the score of an optimal alignment of the i-th and j-th sequences.

2. Use Kruskal's algorithm to find a minimum spanning tree of G. Whenever a minimum spanning tree edge would connect two components, instead add a new root node with directed edges to the roots of the two components. This is the "guide tree".

3. Do pairwise alignments according to the guide tree, working from the leaves to the root. A node u with children v and w corresponds to an alignment of the leaves of v's subtree (already aligned inductively) with the leaves of w's subtree (already aligned).

Details of guide tree construction

1. Initially, each node is the root of its own tree.
2. Consider edges in increasing order of edge label.
3. If the next edge e connects nodes {a,b} in the same tree, discard e.
4. Otherwise, find the root v of the tree containing a, and the root w of the tree containing b. Add a new root u with children v and w, thus merging the trees containing a and b into a single tree.

Details of pairwise alignment

1. Suppose V is an alignment of the sequences at the leaves of v's subtree, and W is an alignment of the sequences at the leaves of w's subtree.
2. Let {a,b} be the pair of sequences that caused these subtrees to be merged, and let A be the optimal alignment of a and b. Use A to guide the alignment of the two alignments V and W.

Complexity analysis

Building the Distance matrix :

- Each pairwise alignment O(n^2)
- Number of pairwise alignments O(k^2)

Iterative construction of MSA

- Number of merge steps O(k)
- Each pairwise alignment O(k^2*n^2)

Entire Algorithm : O(k^2*n^2)

Disadvantages

- The progressive methods are heuristics that are not guaranteed to converge to a global optimum.
- alignment quality can be difficult to evaluate and their true biological significance can be obscure.
- Errors Introduced in steps are propagated and carried over down further, hence making it unreliable.

Conclusion

- Complexity of Smith - Waterman Algorithm : **O (n^k)**
- Progressive alignment construction for multiple sequence alignment: **O (k^2*n^2)**

As we can see the Smith waterman algorithm complexity grows exponentially with increase in no. of DNA sequences. Hence making it highly impractical to use in real life.

The complexity of Progressive alignment construction is polynomial, hence making it feasible to use it. But as discussed it is not that accurate hence it cannot be trusted.

So, we can use Smith waterman algorithm when we have to compare only two strands and Progressive alignment for multiple strands. Another efficient method would be to use Progressive alignment to get a rough idea of a few DNA strands that resemble and then we can verify it using the Smith Waterman Algorithm.