CSE 584 Final Project

Clustering Tools for De Novo mRNA assembly

Shi Shu

Zhijie Qi

Spring 2016

Clustering Tools for De Novo mRNA Assembly

1. Abstract

RNASeq is a powerful technology for directly measuring the frequency of different messenger RNAs among all those being expressed in a population of cells. The most common approach to RNASeq is to align each read back to the genome from which it came. However, if the genome's sequence is not completely known, it may be useful to assemble mRNA sequences directly by piecing together overlapping reads. This is particularly helpful if the goal is to discover novel splice variants of known mRNAs or to find genes that, for whatever reason, may not be represented in your genomic sequence.

Computationally, the first step in mRNA assembly is to divide the reads into clusters according to overlap. If reads S and T overlap by at least a certain number of bases, then they can be clustered together for easier later assembly.

In this paper we will introduce the existing de novo mRNA assembly tools, and implement an algorithm for the overlap detection derived from the Needleman-Wunsch Algorithm, and then implement two clustering algorithm using both hierarchical clustering and k-means clustering for bio sequence using the overlap detection score as the distance function. And test the implemented method with both real world data and synthetic data.

2. Introduction

2.1 De Novo mRNA Assembly

At present, mainly three distinct strategies are applied in short reads assembly. Among them, Greedy-extension is the implementation of string-based method, while De Bruijn graph and overlap-layout-consensus (OLC) are two different graph-based approaches. 1

Overlap graphs are utilized for most assemblers designed for Sanger sequenced reads. The overlaps between each pair of reads is computed and compiled into a graph, in which each node represents a single sequence read. This algorithm is more computationally intensive than de Bruijn graphs, and most effective in assembling fewer reads with a high degree of overlap.2

De Bruijn graphs align k-mers by capturing overlaps of length k-1 between these k-mers and not between the actual reads. The use of k-mers, which are shorter than the read lengths in de Bruijn graphs reduces the computational intensity of the method.[3]

Then we give a brief explanation of two of the popular existing software tools for de novo assembly.

2.1.2 ABySS

ABySS (Assembly By Short Sequences), is a de novo, parallel, paired-end sequence assembler that is designed for short reads. The single-processor version is useful for assembling genomes up to 100 Mbases in size. The parallel version is implemented using MPI and is capable of assembling larger genomes.[4]

Most of the other popular assembly tools like ALLPATHS[5] and Velvet, are all single-threaded applications designed to run on a single processor. However, computation time and memory constraints limit the practical use of these implementations to genomes on the order of a megabase in size.[6] The primary innovation in ABySS is a distributed representation of a de Bruijn graph, which allows parallel computation of the assembly algorithm across a network of commodity computers.

2.1.3 Velvet

Velvet also make use of de Bruijn graphs for genomic sequence assembly. De Bruijn graph is a compact representation based on short words (k-mers) that is ideal for high coverage, very short read data sets. Velvet can be used on very short reads and paired-ends information, one can produce contigs of significant length. The primary innovation of Velvet is that it manipulates the de Bruijn graphs efficiently to both eliminate errors and resolve repeats. These two tasks are done separately: first, the error correction algorithm merges sequences that belong together, then the repeat solver separates paths sharing local overlaps.[7]

2.2 Why is clustering necessary

All assembly tools and algorithm available out there is all very time and space consuming when dealing with large data sets. Then clustering in the pre processing process can be very help. Imaging that instead of assembly 1 million sequence we can assembly 100 different clusters of 10000 sequence. Since all those assembly can be done in parallel and then only an assembly of 100 long sequence is required. Thus properly pre-cluster the input sequences can largely improve the performance of assembly tools. And major clustering tools available for bio sequence is BLASTClust and CD-HIT. And both of them has its own problem. Thus it would be meaningful to develop an accurate clustering tool specifically designed to pre-process the sequence for de novo assembly.

2.3 BLASTClust[8]

BLASTClust is a program within the standalone BLAST package used to cluster either protein or nucleotide sequences. The program begins with pairwise matches and places a sequence in a cluster if the sequence matches at least one sequence already in the cluster.

In the case of proteins, the blastp algorithm is used to compute the pairwise matches; in the case of nucleotide sequences, the Megablast algorithm is used.

The problem with BLASTClust is that it is from the older and deprecated BLAST suite, so it has not been updated for a long time. And a reported bug is that after clustering user would get a very large first cluster and many other small clusters. The result in all the small clusters are accurate but the in the first large cluster , the sequences are not similar at all.

2.4 CD-HIT[9]

CD-HIT is a very widely used program for clustering and comparing protein or nucleotide sequences. CD-HIT was originally developed by Dr. Weizhong Li at Dr. Adam Godzik's Lab at the Burnham Institute.

CD-HIT is very fast and can handle extremely large databases. CD-HIT helps to significantly reduce the computational and manual efforts in many sequence analysis tasks and aids in understanding the data structure and correct the bias within a dataset.

CD-HIT algorithm uses the short word filtering technique. The minimum number of identical short substrings, called 'words', such as dipep- tides, tripeptides and so on, shared by two proteins is a function of their sequence similarity. We calculated this function by analytical and large-scale statistical analyses. Therefore, we can effectively estimate that the similarity of two sequences is below a certain threshold by simple word counting and without an actual sequence alignment.[10]

One of the reported problem with CD-HIT is that the algorithm creates too many clusters in some cases (6000 clusters for 18000 protein sequences[11]). Also even the algorithm can effectively cluster similar sequence together, the result might not be ideal for de novo assembly. Since similar sequence does not necessarily appear in the same region during assembly. While overlapping sequence with starting and ending gap more likely does.

3. Implementation

The project was first written in Matlab due to the build in clustering function and the optimized matrix manipulation for overlap detection. But later on we realized that a suffix tree implementation is required. And implementing a suffix tree in Matlab would be very inefficient since not much build in functions can be utilized and would generate huge cost to the performance of the clustering algorithm. Then the project is all migrated to python.

3.1 Overlap detection

The core idea of the designed clustering algorithm is that using standard clustering method but replace the distance function with the designed overlap detection function.

3.1.1 Needleman-Wunsch Algorithm

The overlap detection function is derived from the standard Needleman-Wunsch Algorithm for global alignment.

Let's say we want to perform a global alignment on two strings S and T. Let's say S is of length m and T is of length n. And we can build up a scoring matrix through the following rules.

Let's call the scoring matrix M.

M[i,j] denote the score of the best alignment score of S[1,…i] and T[1,…j].

1. Initialize the scoring matrix, M[0,0] = 0.

2. Then we fill out the matrix using the following formula. Here we do not consider the gap opening penalty and gap extending penalty. All gaps have a score of -1. And the part ((S[i] == T[j])*2-1) will equal to 1 if S[i] = T[j] and -1 otherwise.

$$M[i,j] = max\begin{cases} M[i-1,j-1] + ((S[i]==T[j])*2-1) \\ M[i-1,j] - 1 \\ M[i,j-1] - 1 \end{cases}$$

3. Return the score at M[m,n]. And the for the purpose of the project. We do not concerns about the route of the best alignment. We only care about the best alignment score.

3.1.2 Overlap detection

Then to derive the overlap detection algorithm. First, we need to find the changes that we need to make to the Needleman-Wunsch Algorithm. The goal of the overlap detection algorithm is to ignore all the opening and closing gaps and only care about the the gap and mismatch happens in the middle.

Thus we have our overlap detection algorithm as following.

1. Initialize the scoring matrix, M[:,0] = 0 and M[0,:] = 0. Here M[:,i] means the ith column of the matrix M and M[i,:] means the ith row of the matrix M.

2. Then we fill out the matrix using the same following formula.

$$M[i,j] = max\begin{cases} M[i-1,j-1] + ((S[i]==T[j])*2-1) \\ M[i-1,j] - 1 \\ M[i,j-1] - 1 \end{cases}$$

3. Find the max(M[m,:]) and max(M[:,n]). Then return the max of those two numbers as the result.

3.2 Filtering before the overlap detection

After testing the above algorithm with Hierarchical clustering algorithm in Matlab. We soon found out that this is too slow to let it run on any large set of sequences. Due to the fact that the alignment process has complexity $O(n^2)$. But the fact is many sequence is not similar at all hence should not be clustered together after first glance. Thus we moved on to implement a pre filtering process for the overlap detection function.

One straight forward method to filter two input sequence is check the length of the longest common subsequence of those two sequence and if that length is lower than a threshold, we can kill the process and does not continue to the overlap detection process.

Thus a linear time function to find the longest common subsequence is implemented as the filter.

This would bring the cost of the most of the pair wise comparison to linear time and only some would have to go through the quadratic time overlap detection process.

3.2.1 Longest Common Subsequence

In order to implement linear longest common subsequence solver, we need a linear time suffix tree construction. And then use the algorithm to build a generalized suffix tree for both input sequence S and T. And then find the longest common subsequence in linear time.

The algorithm that we use here to construct the suffix tree is Ukkonen's algorithm.[12]

To construct a suffix tree for string S of length m.

```
For i from 1 to m
    For j from 1 to i+1
        Find the end of the path from the root labelled S[j..i] in the current tree.
        Extend that path by adding character S[i+l] if it is not there already[13]
```

Even thought the algorithm does not look like a linear time construction. Proof of the algorithm is linear time can be found in the original Ukkonen's paper in reference [12].

Then with the linear suffix tree construction. To calculate the longest common subsequence for S and T, we construct a new generalized suffix tree for the string S#T$, where # and $ does not appear in S or T. Then in the suffix tree of S#T$, for every internal node, it has three different situation.

1. The internal node have leaves from S only. Then we label the internal node S.

2. The internal node have leaves from T only. Then we label the internal node T.
3. The internal node have leaves from both S and T. Then we label the internal node ST.

Find the deepest node that is labelled ST. And return length of the string from root to this internal node.

Now with the longest common subsequence, we can modify our overlap detection function to use the longest common subsequence to first filter the input. If the inputs has a LCS longer than the threshold k, we proceed to alignment process, otherwise return the score of -sys.maxint as the result in python.

3.3 Clustering Tools

3.3.1 K-medoids Clustering

Applying overlap score as distance metric, ‘k-medoids’ is used as one clustering method to assign sequences to different clusters.’k-medoids’ is similar to ‘k-means’, but instead of manually computing the coordinates of the cluster center, it uses one existing data points from the dataset as the cluster center. Applied to strings in our case, using ‘k-medoids’ avoids the complexity of transferring strings to numerical values.

The algorithm of ‘k-medoids’ we used here is described below:

```
Randomly select k of the n data points as the medoids
Associate each data point to the closest medoid.
While at least one of the medoids is changing and iteration is less than maximum iterations:
        For each medoid m, for each non-medoid data point o:
                Swap m and o, recompute the sum of distances of points to the medoid
                If the sum increased in the previous step, undo the swap
Return data point cluster labels
```

Regarding choosing the value of k, there are two ways to do it. For small datasets, the elbow method is used. Different values of k are iterated and for each value of k, the cost of resulting clustering solution is computed. Then we plot the cost as a function of k, as shown in Figure 1, and pick the elbow point as our choice of k. If the dataset is too large to run k-medoids multiple times, we use equation 1 as the ‘Rule of Thumb’ to get the value of k.[14]

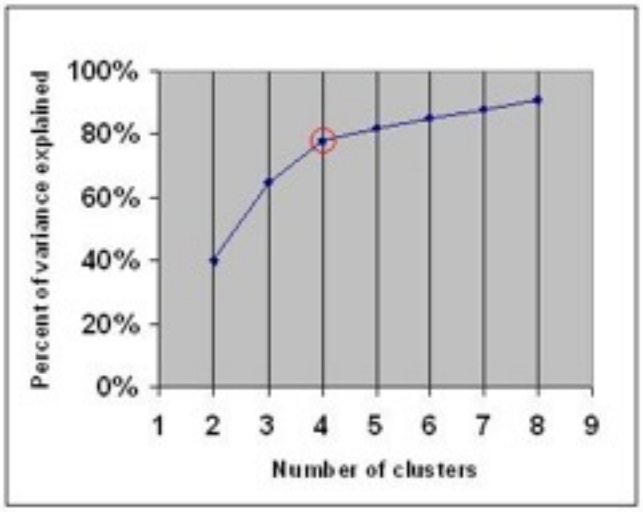


Figure 1: Score as a function of number of clusters[15]

$$k = \lceil \frac{\sqrt{\#dataPoints}}{2} \rceil \tag{1}$$

3.3.2 Hierarchical Clustering

Another clustering method we used is hierarchical clustering. Linkage tree is first built using overlap score as the distance metric. Then a dendrogram is constructed based on the linkage tree. The dendrogram can be truncated according to the number of clusters we want and corresponding cluster labels can be assigned to the leaves after the truncation. The order of leaves is extracted from the dendrogram and is used to rearrange the original distance matrix so that similar sequences will be put close to each other. Finally a heat map is plotted based on the rearranged matrix. From analyzing the diagonal regions of the heat map, we are able to visualize the distinctions between different clusters.

4. Evaluation

To evaluate the validness of our clustering methods, we first test them with a small test case.

A 500-character string is manually generated to represent the original assembly. Then 500 50-character substrings are sampled with replacement from the original string to represent the rnaSeq data. Each substring is labeled with its position in the original string, which is to be used for further computation. Based on equation ?, the number of clusters is chosen to be 12. Then k-medoids and hierarchical clustering are both applied to the mini rnaSeq data. After getting the clustering results, for each cluster, the mean and standard deviation of the position indices are computed. The expected results are that the mean positions should be away from each other and the stand deviations of each cluster should be small. The test results of both methods and their comparison are listed in Table 1, 2 and 3 respectively.

From Table 3, we can see that both clustering results have mean positions at least 19 characters away each other and have a nice coverage all over the original string. The standard deviation of positions within the same cluster ranges from 4.30 to 18.77 for k-Medoids and from 5.23 to 17.94 for hierarchical. With the number of cluster equal to 12, the dendrogram of hierarchical clustering is partitioned and colored, as shown in Figure ?. A heat map is also generated. From Figure 2, we are able to visualize that different clusters are well distinct from each other.

All these results show that substrings from similar positions are clustered together and both methods yield similar clustering results. As a result, both k-Medoids and hierarchical clustering using overlap scoring as distance metric are valid clustering methods in this case.

Table 1 k-Medoids Clustering Result Analysis

Cluster Label	Mean Position	STD of position
1	6.681818182	4.29996156
2	436.7142857	7.987745717
3	230.3448276	14.27507078
4	361.2909091	14.77180696
5	293.3703704	18.77313844
6	108.1176471	10.04040281
7	76.03846154	7.708540564
8	50.06666667	6.687965975
9	405.3571429	9.423429334
10	144.8536585	10.51029919
11	184.1458333	12.5399189
12	26.21428571	6.172271768

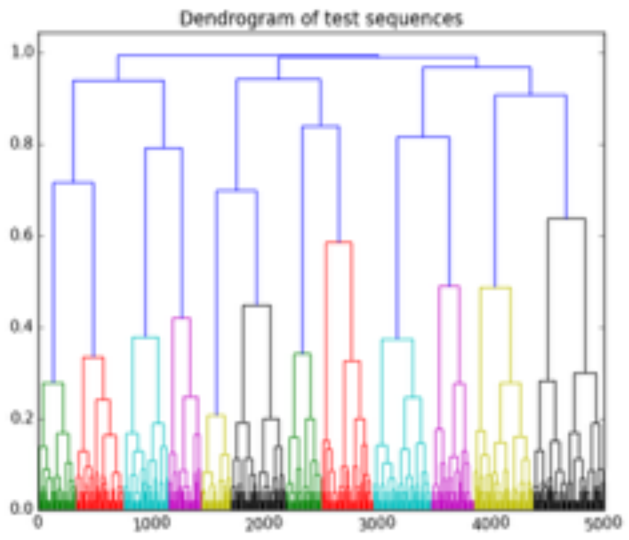
Table 2 Hierarchical Clustering Result Analysis

Cluster Label	Mean Position	STD of position
1	5.35172828	5.23426816
2	27.632241	7.325614
3	52.365413	6.0404623

4	76.03846154	6.8563165
5	108.1176471	10.04040281
6	136.231546	10.51029919
7	190.256847	13.26584862
8	238.2195122	13.75325846
9	290.3615687	17.9365246
10	350.2684162	16.32154445
11	398.365827	10.02230441
12	441.235619	7.716577742

Table 3. Comparison of k-medoids and hierarchical clustering results

	k-Medoids	**Hierarchical**
Min Position Interval	19.53	22.18
Position Span	6.68-436.35	5.35-441.24
Min Position Std	4.30	5.23
Max Position Std	18.77	17.94



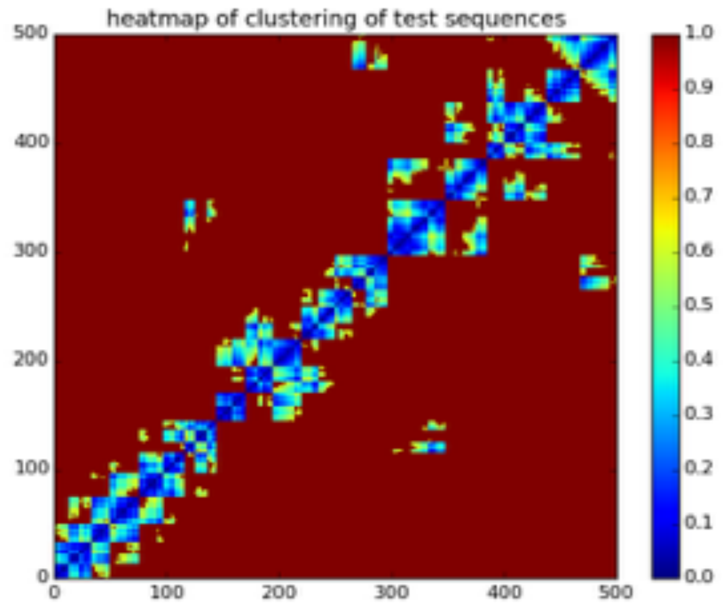


Figure 2: Dendrogram and Heat map of based on hierarchical clustering of test sequences

The whole genome of Acinetobacter phage YMC13/03/R2096[16] is used to test our clustering methods in a 'real world' environment. The phage has a scaffold genome length of 98,170 base pairs. The rnaSeq data generated consists of 70,000 sub-sequences, each with 76 bps. After computing out the distance matrix, both k-Medoids and hierarchical clustering were operated. While k-Medoids successfully terminated after several hours, hierarchical clustering failed to generate the output due to a hardware limitation error of "maximum depth of recursion exceeded" happening at computing the dendrogram. The summary of k-Medoids clustering solutions and its comparison to the toy test case above are listed in Table 4.
Table 4 shows that the resulting clusters a nice coverage over the genome and cluster centers are away from each other. Although compared with the toy test case, the position standard deviation increases, this is caused by the less coverage of rnaSeq data over the original genome and the maximum standard deviation is still within the range of half sub-sequence length, which is 38 bp in this case. These results show that the clustering results are trustworthy and k-Medoids clustering method is scalable to large or even larger dataset.
Regarding the failure of hierarchical clustering on large dataset, it is caused by the hardware limitation of our own computers. With increased computing power, it is believed that hierarchical clustering will also produce promising solutions.

Table 4. Summary of k-Medoids clustering on Exophiala *aquamarina* rnaSeq data

	Exophiala *aquamarina*	**Toy test case**
Min Position Interval	29.52	19.53

Position Span	11.23-98145.52	6.68-436.35
Min Position Std	6.55	4.30
Max Position Std	30.26	18.77

5. Conclusion

In conclusion, the project implemented an overlap detection with longest common subsequence as a filter and apply two different clustering algorithm using overlap detection as the distance function and test the accuracy of the result.

Both real world cases and synthetic cases show that when performing clustering as a pre processing process for de novo assembly, overlap detection is a decent criteria to be used as the distance function.

Due to the time limitation, we did not use the de novo assembly software tools to assemble the result from clustering. With proper hardware and enough time, we would feed the clustering test result to the assembly software and then we would compare the assembled result with the original genome and further validate the accuracy of the clustering.

6. Reference

[1] Zhang W, Chen J, Yang Y, Tang Y, Shang J, Shen B. A Practical Comparison of De Novo Genome Assembly Software Tools for Next-Generation Sequencing Technologies. Jordan IK, ed. PLoS ONE. 2011;6(3):e17915. doi:10.1371/journal.pone.0017915.

[2] Illumina, Inc. (2010). "De Novo Assembly Using Illumina Reads"

[3] Zerbino DR, Birney E (2008) Velvet: algorithms for *de novo* short read as- sembly using de Bruijn graphs. Genome Research 18: 821–829.

[4] http://www.bcgsc.ca/platform/bioinfo/software/abyss

[5] Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I.A., Belmonte, M.K., Lander, E.S., Nusbaum, C., Jaffe, D.B.(2008) ALLPATHS: De novo assembly of whole-genome shotgun microreads. Genome Res.

[6] Simpson, Jared T., et al. "ABySS: a parallel assembler for short read sequence data." Genome research 19.6 (2009): 1117-1123.

[7] Zerbino, Daniel R., and Ewan Birney. "Velvet: algorithms for de novo short read assembly using de Bruijn graphs." Genome research 18.5 (2008): 821-829.

[8] http://www.ncbi.nlm.nih.gov/Web/Newsltr/Spring04/blastlab.html

[9] http://weizhongli-lab.org/cd-hit/

[10] Li, Weizhong, and Adam Godzik. "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences." Bioinformatics 22.13 (2006): 1658-1659.

[11] https://www.biostars.org/p/154126/

[12] Ukkonen, Esko. "On-line construction of suffix trees." Algorithmica 14.3 (1995): 249-260.

[13] Gusfield, Dan. Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge university press, 1997.

[14] Cyril Goutte, Lars Kai Hansen, Matthew G. Liptrot & Egill Rostrup (2001). "Feature-Space Clustering for fMRI Meta-Analysis". Human Brain Mapping 13 (3): 165–183.doi:10.1002/hbm.1031. PMID 11376501

[15] H.S. Park , C.H. Jun, A simple and fast algorithm for K-medoids clustering, Expert Systems with Applications, 36, (2) (2009), 3336–3341.

[16] http://www.ebi.ac.uk/ena/data/view/KM672662&display=txt&expanded=true